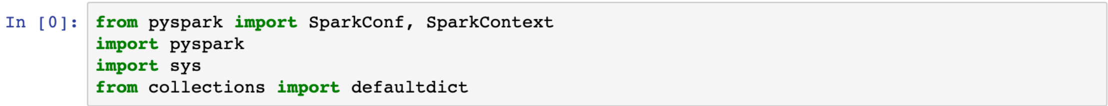
EECS E6893 HW2

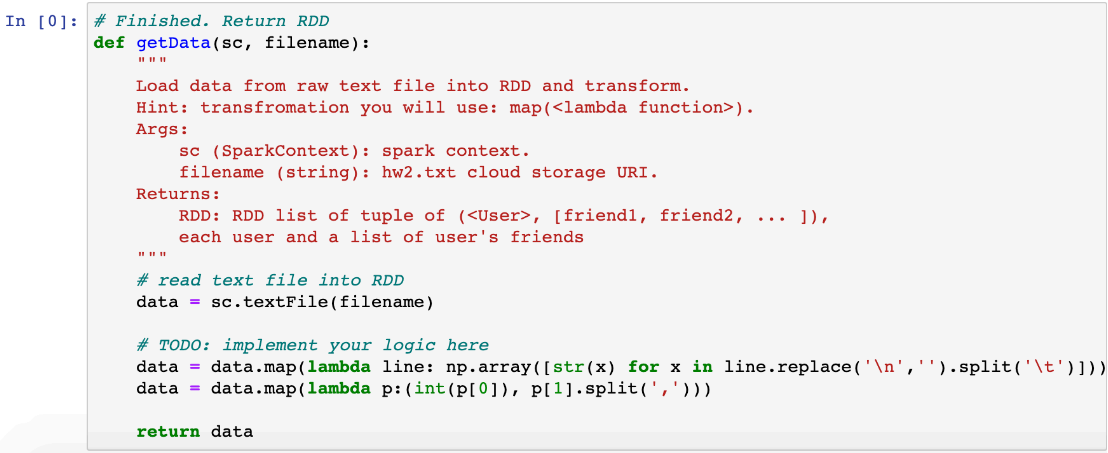
Jing Qian (jq2282)

Question 1

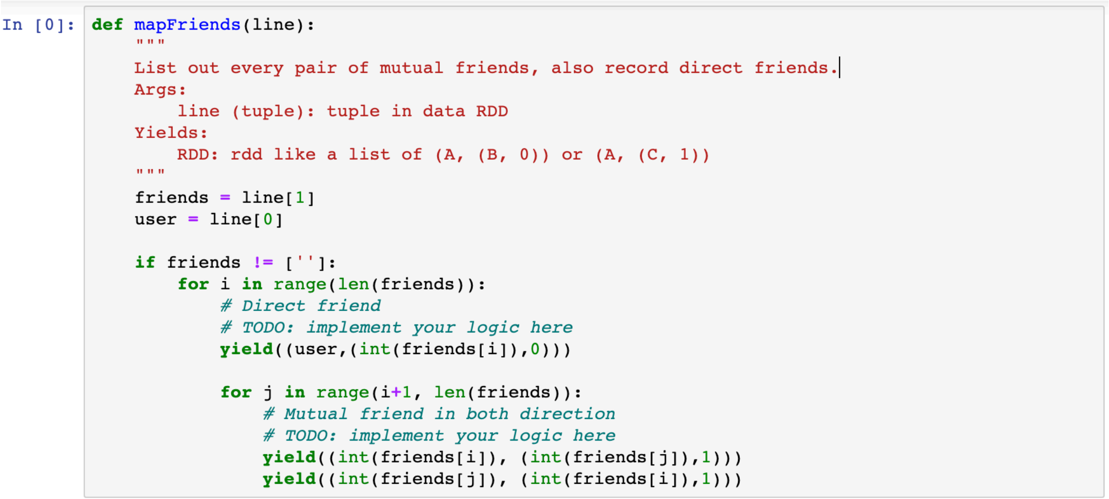
(1) Screenshots of the code



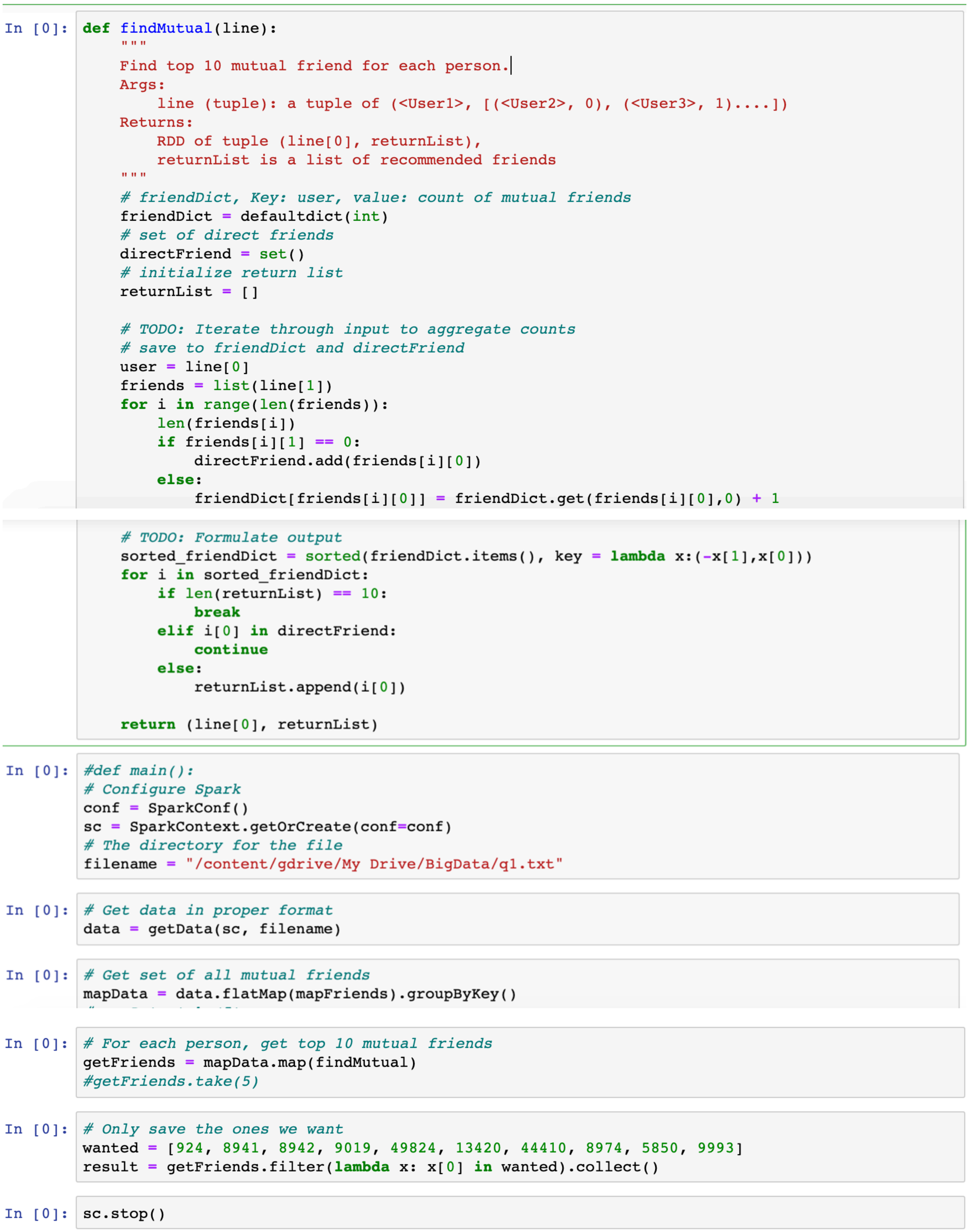
```
In [0]: from pyspark import SparkConf, SparkContext
        import pyspark
        import sys
        from collections import defaultdict
```



```
In [0]: # Finished. Return RDD
        def getData(sc, filename):
            """
            Load data from raw text file into RDD and transform.
            Hint: transfromation you will use: map(<lambda function>).
            Args:
                sc (SparkContext): spark context.
                filename (string): hw2.txt cloud storage URI.
            Returns:
                RDD: RDD list of tuple of (<User>, [friend1, friend2, ... ]),
                each user and a list of user's friends
            """
            # read text file into RDD
            data = sc.textFile(filename)

            # TODO: implement your logic here
            data = data.map(lambda line: np.array([str(x) for x in line.replace('\n','').split('\t')]))
            data = data.map(lambda p:(int(p[0]), p[1].split(',')))

            return data
```



```
In [0]: def mapFriends(line):
            """
            List out every pair of mutual friends, also record direct friends.
            Args:
                line (tuple): tuple in data RDD
            Yields:
                RDD: rdd like a list of (A, (B, 0)) or (A, (C, 1))
            """
            friends = line[1]
            user = line[0]

            if friends != ['']:
                for i in range(len(friends)):
                    # Direct friend
                    # TODO: implement your logic here
                    yield((user,(int(friends[i]),0)))

                    for j in range(i+1, len(friends)):
                        # Mutual friend in both direction
                        # TODO: implement your logic here
                        yield((int(friends[i]), (int(friends[j]),1)))
                        yield((int(friends[j]), (int(friends[i]),1)))
```



```python
def findMutual(line):
    """
    Find top 10 mutual friend for each person.
    Args:
        line (tuple): a tuple of (<User1>, [(<User2>, 0), (<User3>, 1)....])
    Returns:
        RDD of tuple (line[0], returnList),
        returnList is a list of recommended friends
    """
    # friendDict, Key: user, value: count of mutual friends
    friendDict = defaultdict(int)
    # set of direct friends
    directFriend = set()
    # initialize return list
    returnList = []

    # TODO: Iterate through input to aggregate counts
    # save to friendDict and directFriend
    user = line[0]
    friends = list(line[1])
    for i in range(len(friends)):
        len(friends[i])
        if friends[i][1] == 0:
            directFriend.add(friends[i][0])
        else:
            friendDict[friends[i][0]] = friendDict.get(friends[i][0],0) + 1

    # TODO: Formulate output
    sorted_friendDict = sorted(friendDict.items(), key = lambda x:(-x[1],x[0]))
    for i in sorted_friendDict:
        if len(returnList) == 10:
            break
        elif i[0] in directFriend:
            continue
        else:
            returnList.append(i[0])

    return (line[0], returnList)
```

```python
#def main():
# Configure Spark
conf = SparkConf()
sc = SparkContext.getOrCreate(conf=conf)
# The directory for the file
filename = "/content/gdrive/My Drive/BigData/q1.txt"
```

```python
# Get data in proper format
data = getData(sc, filename)
```

```python
# Get set of all mutual friends
mapData = data.flatMap(mapFriends).groupByKey()
```

```python
# For each person, get top 10 mutual friends
getFriends = mapData.map(findMutual)
#getFriends.take(5)
```

```python
# Only save the ones we want
wanted = [924, 8941, 8942, 9019, 49824, 13420, 44410, 8974, 5850, 9993]
result = getFriends.filter(lambda x: x[0] in wanted).collect()
```

```python
sc.stop()
```

(2) Screenshots of the recommendation results.

In [12]:
```
for i in sorted(result,key = lambda x:x[0]):
    print(i)
```

```
(924, [439, 2409, 6995, 11860, 15416, 43748, 45881])
(5850, [5819, 5805, 5811, 5815, 5828, 5831, 5836, 219, 576, 639])
(8941, [8943, 8944, 8940])
(8942, [8939, 8940, 8943, 8944])
(8974, [8960, 12241, 8774, 6973, 8969, 8980, 8982, 8984, 8978, 8979])
(9019, [9022, 317, 9023])
(9993, [9991, 13134, 13478, 13877, 34299, 34485, 34642, 37941])
(13420, [4736, 7651, 10469, 14264, 351, 2101, 2554, 7608, 8508, 8711])
(44410, [4231, 44462, 351, 4302, 6318, 8221, 9095, 10328, 10370, 10462])
(49824, [49846, 41581, 43382, 49786, 49788, 49789, 49814, 49819, 49834, 16])
```

Question 2 Graph Analysis

In [7]:
```
import numpy as np
from pyspark import *
```

In [8]:
```
# Configure Spark
conf = SparkConf()
sc = SparkContext.getOrCreate(conf=conf)
# The directory for the file
filename = "q1.txt"
```

In [36]:
```
# Finished. Return RDD
def getData(sc, filename):
    """
    Load data from raw text file into RDD and transform.
    """
    # read text file into RDD
    data = sc.textFile(filename)

    # TODO: implement your logic here
    data = data.map(lambda line: np.array([str(x) for x in line.replace('\n','').split('\t')]))
    data = data.map(lambda p:(int(p[0]), p[1].split(',')))

    return data
```

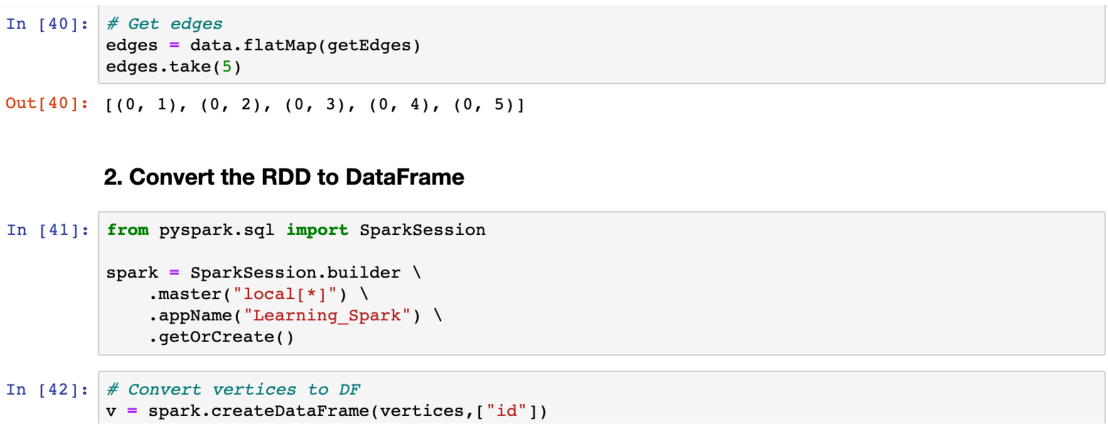
In [37]:
```
def getEdges(line):
    # similar to mapFriends() in Q1, edges are direct friendship
    friends = line[1]
    user = line[0]

    if friends != ['']:
        for i in range(len(friends)):
            # Direct friend
            yield((user, int(friends[i])))
```

1. Format data into edges and vertices

In [38]:
```
# Get data in proper format
data = getData(sc, filename)
```

In [39]:
```
# Get vertics
vertices = data.map(lambda x: (x[0],))
vertices.take(5)
```

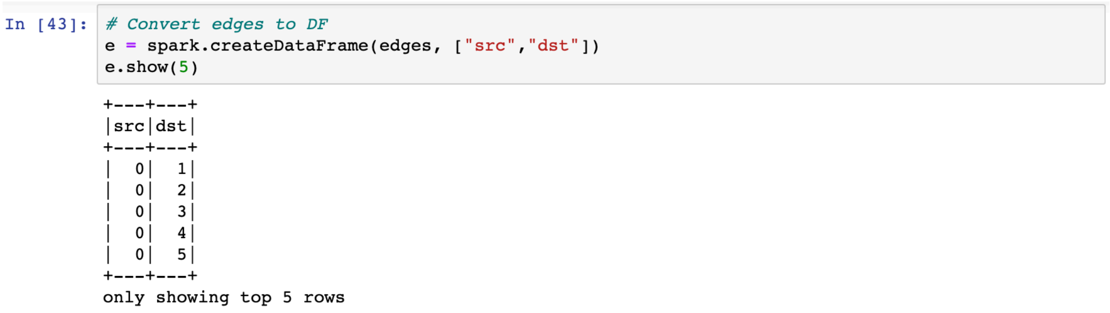
Out[39]:
```
[(0,), (1,), (2,), (3,), (4,)]
```



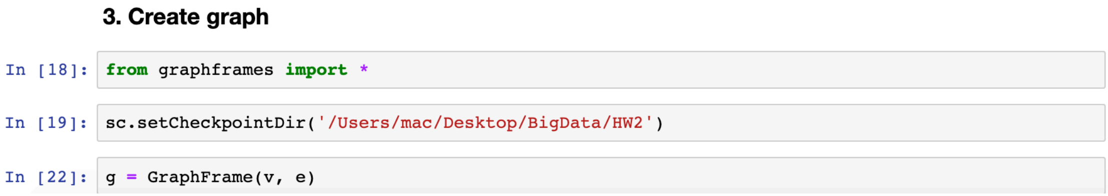
```
In [40]: # Get edges
         edges = data.flatMap(getEdges)
         edges.take(5)

Out[40]: [(0, 1), (0, 2), (0, 3), (0, 4), (0, 5)]
```

2. Convert the RDD to DataFrame

```
In [41]: from pyspark.sql import SparkSession

         spark = SparkSession.builder \
             .master("local[*]") \
             .appName("Learning_Spark") \
             .getOrCreate()

In [42]: # Convert vertices to DF
         v = spark.createDataFrame(vertices,["id"])
```

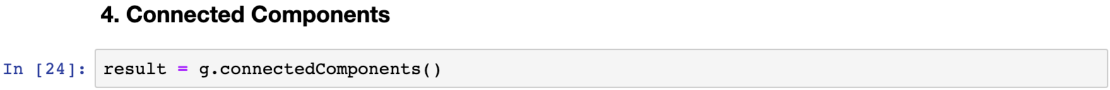


```
In [43]: # Convert edges to DF
         e = spark.createDataFrame(edges, ["src","dst"])
         e.show(5)

         +---+---+
         |src|dst|
         +---+---+
         |  0|  1|
         |  0|  2|
         |  0|  3|
         |  0|  4|
         |  0|  5|
         +---+---+
         only showing top 5 rows
```



3. Create graph

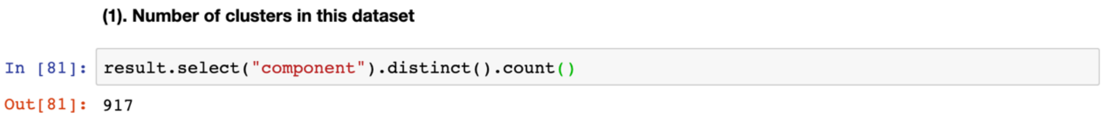
```
In [18]: from graphframes import *

In [19]: sc.setCheckpointDir('/Users/mac/Desktop/BigData/HW2')

In [22]: g = GraphFrame(v, e)
```



4. Connected Components

```
In [24]: result = g.connectedComponents()
```

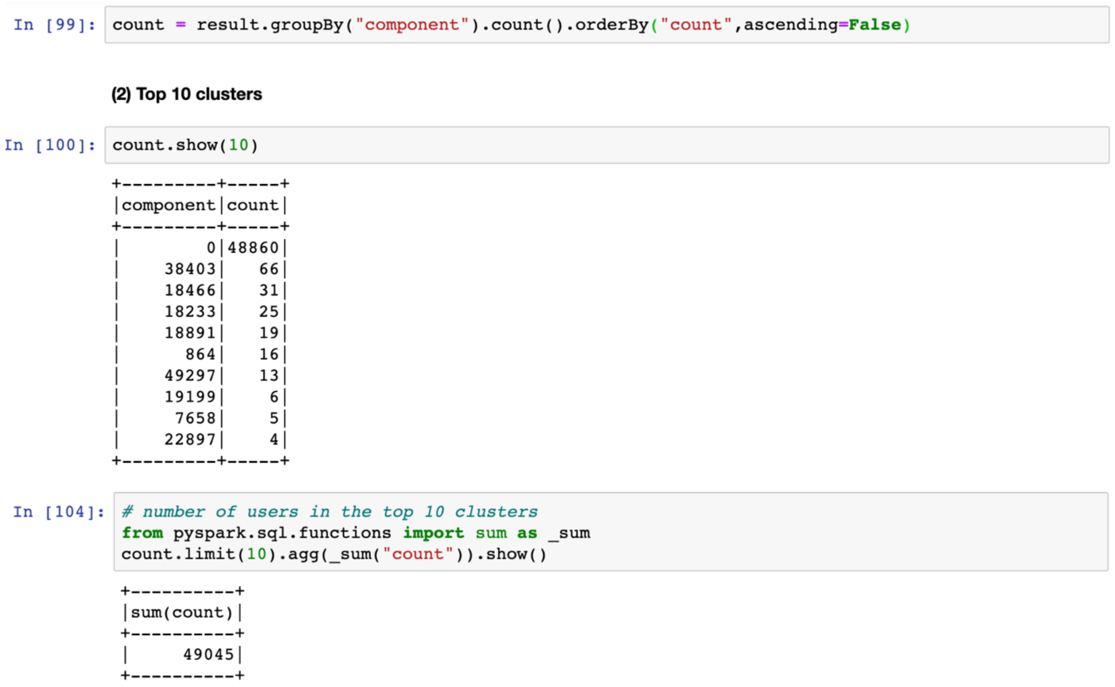
(1) There are 917 clusters/connected components in total for this dataset.



(1). Number of clusters in this dataset

```
In [81]: result.select("component").distinct().count()

Out[81]: 917
```

(2) There are 49045 users in the top 10 clusters.



```
In [99]: count = result.groupBy("component").count().orderBy("count",ascending=False)
```

(2) Top 10 clusters

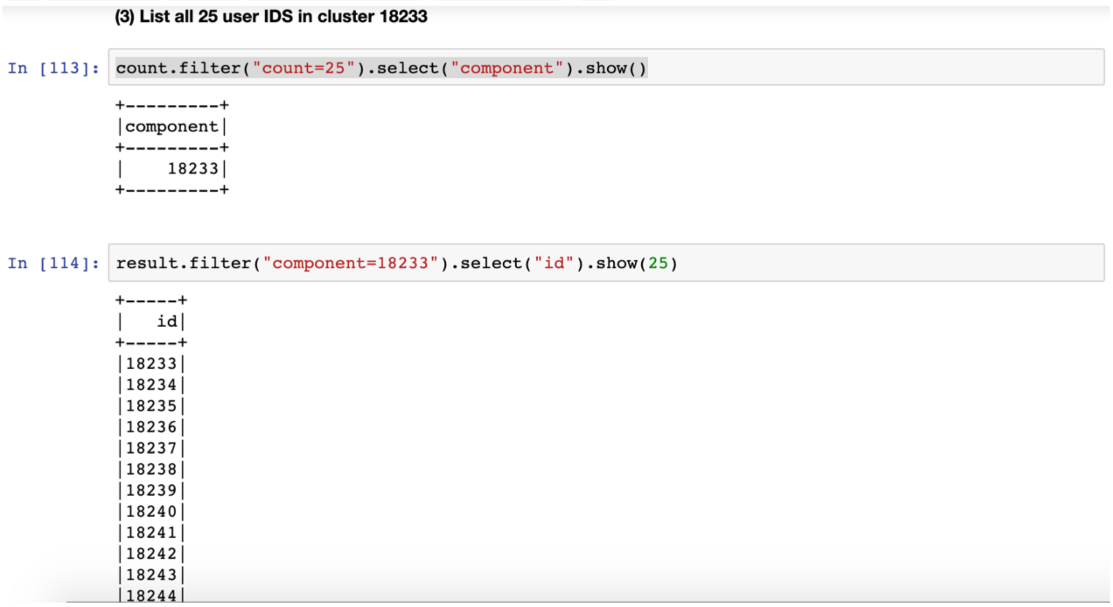
```
In [100]: count.show(10)
```

```
+---------+-----+
|component|count|
+---------+-----+
|        0|48860|
|    38403|   66|
|    18466|   31|
|    18233|   25|
|    18891|   19|
|      864|   16|
|    49297|   13|
|    19199|    6|
|     7658|    5|
|    22897|    4|
+---------+-----+
```

```
In [104]: # number of users in the top 10 clusters
          from pyspark.sql.functions import sum as _sum
          count.limit(10).agg(_sum("count")).show()
```

```
+----------+
|sum(count)|
+----------+
|     49045|
+----------+
```

(3) The user ids for the cluster which has 25 users are: 18233 - 18257.



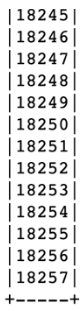
(3) List all 25 user IDS in cluster 18233

```
In [113]: count.filter("count=25").select("component").show()
```

```
+---------+
|component|
+---------+
|    18233|
+---------+
```

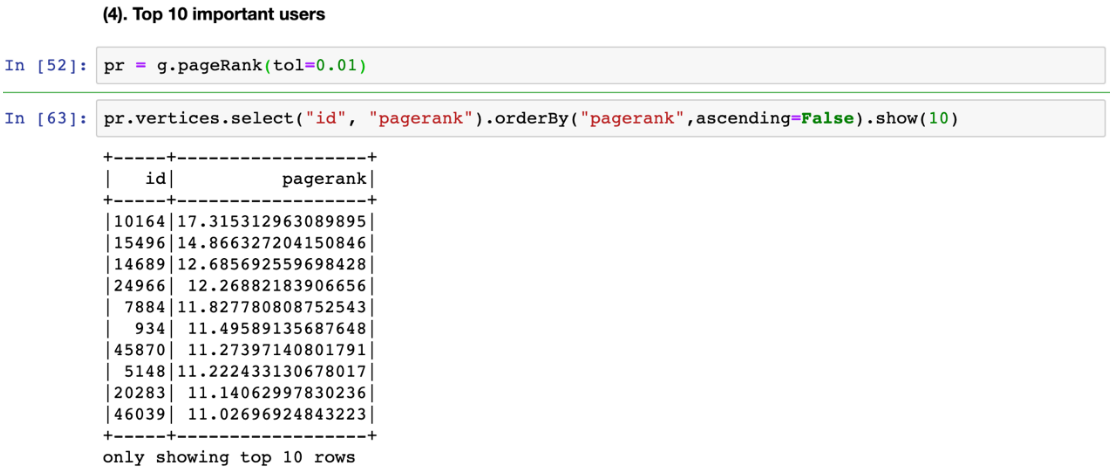
```
In [114]: result.filter("component=18233").select("id").show(25)
```

```
+-----+
|   id|
+-----+
|18233|
|18234|
|18235|
|18236|
|18237|
|18238|
|18239|
|18240|
|18241|
|18242|
|18243|
|18244|
```



```
|18245|
|18246|
|18247|
|18248|
|18249|
|18250|
|18251|
|18252|
|18253|
|18254|
|18255|
|18256|
|18257|
+-----+
```

(4) The most important user is the one with User ID 10164.



(4). Top 10 important users

```
In [52]: pr = g.pageRank(tol=0.01)
```

```
In [63]: pr.vertices.select("id", "pagerank").orderBy("pagerank",ascending=False).show(10)
```

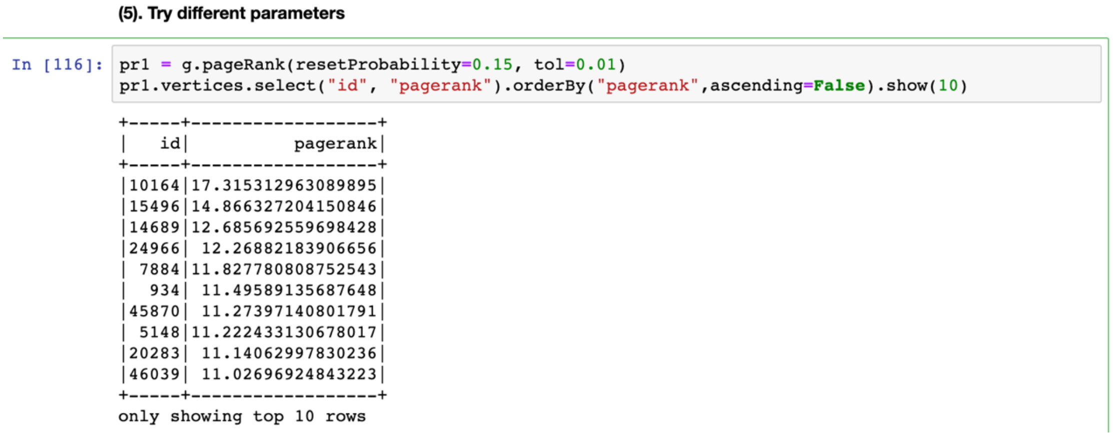
```
+-----+------------------+
|   id|          pagerank|
+-----+------------------+
|10164|17.315312963089895|
|15496|14.866327204150846|
|14689|12.685692559698428|
|24966| 12.26882183906656|
| 7884|11.827780808752543|
|  934| 11.49589135687648|
|45870| 11.27397140801791|
| 5148|11.222433130678017|
|20283| 11.14062997830236|
|46039| 11.02696924843223|
+-----+------------------+
only showing top 10 rows
```

(5) Using different parameters setting for PageRank would lead to differences in the result.

* Increase the **"resetProbability"** from 0.15 (pr1) to 0.5 (pr2). We could see that the top four important users remain the same and the rest are different. "resetProbability" is the parameter that defines probability of resetting to a random vertex.

* Increase the **"tol"** from 0.01 (pr1) to 0.1 (pr3). We could see that the top four important users remain the same and the rest are different. "tol" is the parameter that defines the convergence tolerance that algorithm runs. Increasing "tol" tends to decrease the algorithm iteration numbers.

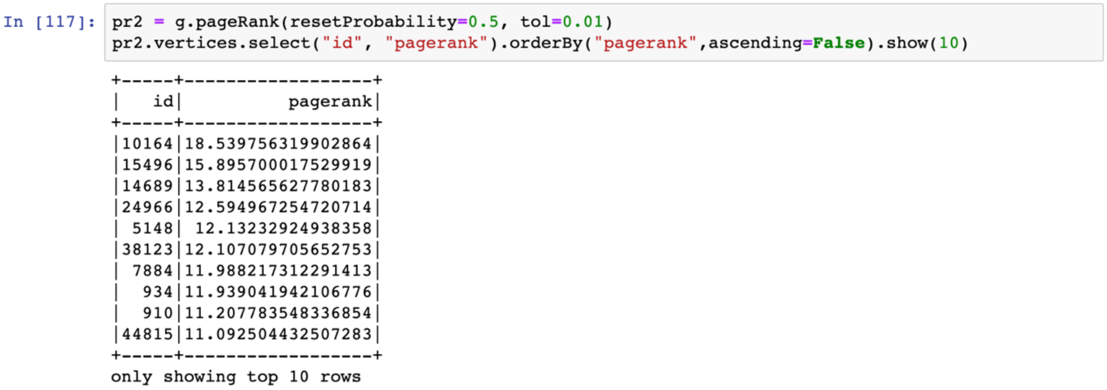
* Set the **"sourceId"** to 10164 (pr4). We could see that only the top user 10164 remains its position while others all change. Also, the pagerank values are quite different from original ones. "sourceId" is the parameter that assigns the source vertex for a personalized PageRank.



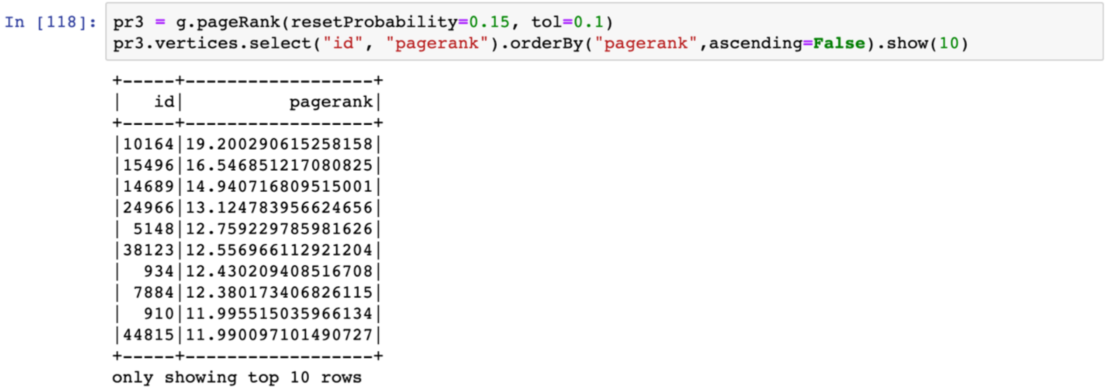
(5). Try different parameters

```
In [116]: pr1 = g.pageRank(resetProbability=0.15, tol=0.01)
          pr1.vertices.select("id", "pagerank").orderBy("pagerank",ascending=False).show(10)
```

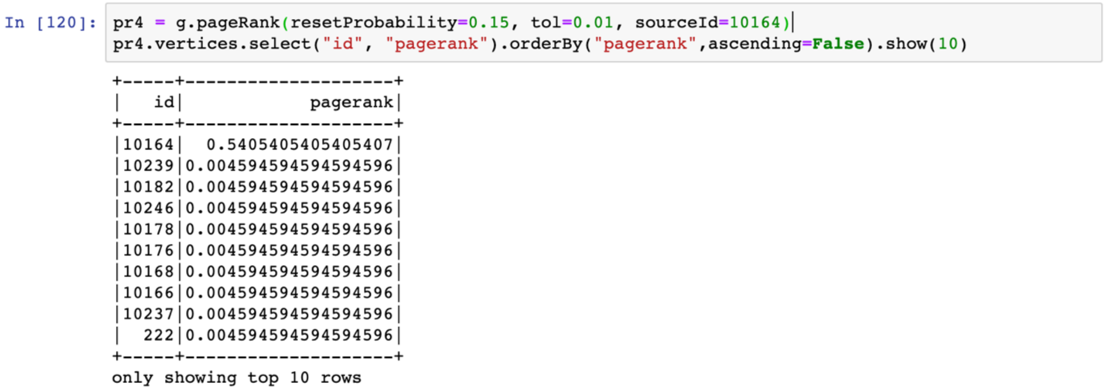
```
+-----+------------------+
|   id|          pagerank|
+-----+------------------+
|10164|17.315312963089895|
|15496|14.866327204150846|
|14689|12.685692559698428|
|24966| 12.26882183906656|
| 7884|11.827780808752543|
|  934| 11.49589135687648|
|45870| 11.27397140801791|
| 5148|11.222433130678017|
|20283| 11.14062997830236|
|46039| 11.02696924843223|
+-----+------------------+
only showing top 10 rows
```



```
In [117]: pr2 = g.pageRank(resetProbability=0.5, tol=0.01)
          pr2.vertices.select("id", "pagerank").orderBy("pagerank",ascending=False).show(10)

          +-----+------------------+
          |   id|          pagerank|
          +-----+------------------+
          |10164|18.539756319902864|
          |15496|15.895700017529919|
          |14689|13.814565627780183|
          |24966|12.594967254720714|
          | 5148| 12.13232924938358|
          |38123|12.107079705652753|
          | 7884|11.988217312291413|
          |  934|11.939041942106776|
          |  910|11.207783548336854|
          |44815|11.092504432507283|
          +-----+------------------+
          only showing top 10 rows
```

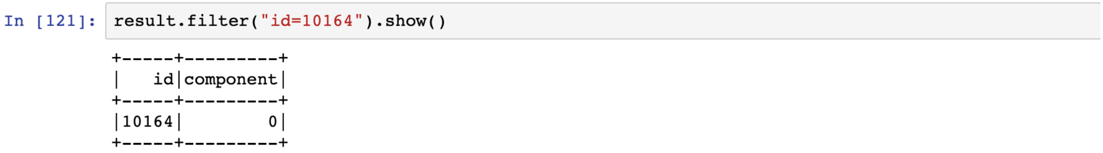


```
In [118]: pr3 = g.pageRank(resetProbability=0.15, tol=0.1)
          pr3.vertices.select("id", "pagerank").orderBy("pagerank",ascending=False).show(10)

          +-----+------------------+
          |   id|          pagerank|
          +-----+------------------+
          |10164|19.200290615258158|
          |15496|16.546851217080825|
          |14689|14.940716809515001|
          |24966|13.124783956624656|
          | 5148|12.759229785981626|
          |38123|12.556966112921204|
          |  934|12.430209408516708|
          | 7884|12.380173406826115|
          |  910|11.995515035966134|
          |44815|11.990097101490727|
          +-----+------------------+
          only showing top 10 rows
```

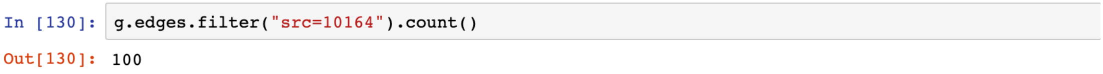


```
In [120]: pr4 = g.pageRank(resetProbability=0.15, tol=0.01, sourceId=10164)
          pr4.vertices.select("id", "pagerank").orderBy("pagerank",ascending=False).show(10)

          +-----+--------------------+
          |   id|            pagerank|
          +-----+--------------------+
          |10164|  0.5405405405405407|
          |10239|0.004594594594594596|
          |10182|0.004594594594594596|
          |10246|0.004594594594594596|
          |10178|0.004594594594594596|
          |10176|0.004594594594594596|
          |10168|0.004594594594594596|
          |10166|0.004594594594594596|
          |10237|0.004594594594594596|
          |  222|0.004594594594594596|
          +-----+--------------------+
          only showing top 10 rows
```

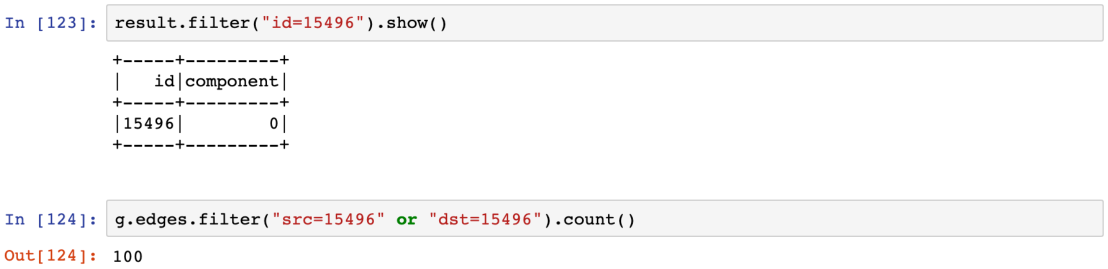
(6) The user with user id 10164 is the most important user. It belongs to the main cluster -- component 0, which is much larger than the rest clusters. Moreover, our graph is a bidirection graph. User 10164 has 100 edges, which means s/he is friends with many people and is able to share their importance. The second important user has the similar properties. These properties, being in the main cluster and having many edges, make 10164 the most important user.



```
In [121]: result.filter("id=10164").show()

          +-----+---------+
          |   id|component|
          +-----+---------+
          |10164|        0|
          +-----+---------+
```



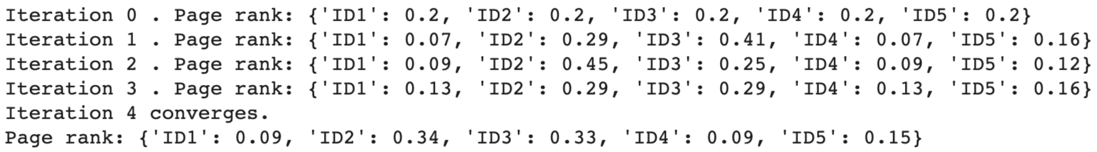
```
In [130]: g.edges.filter("src=10164").count()
Out[130]: 100
```



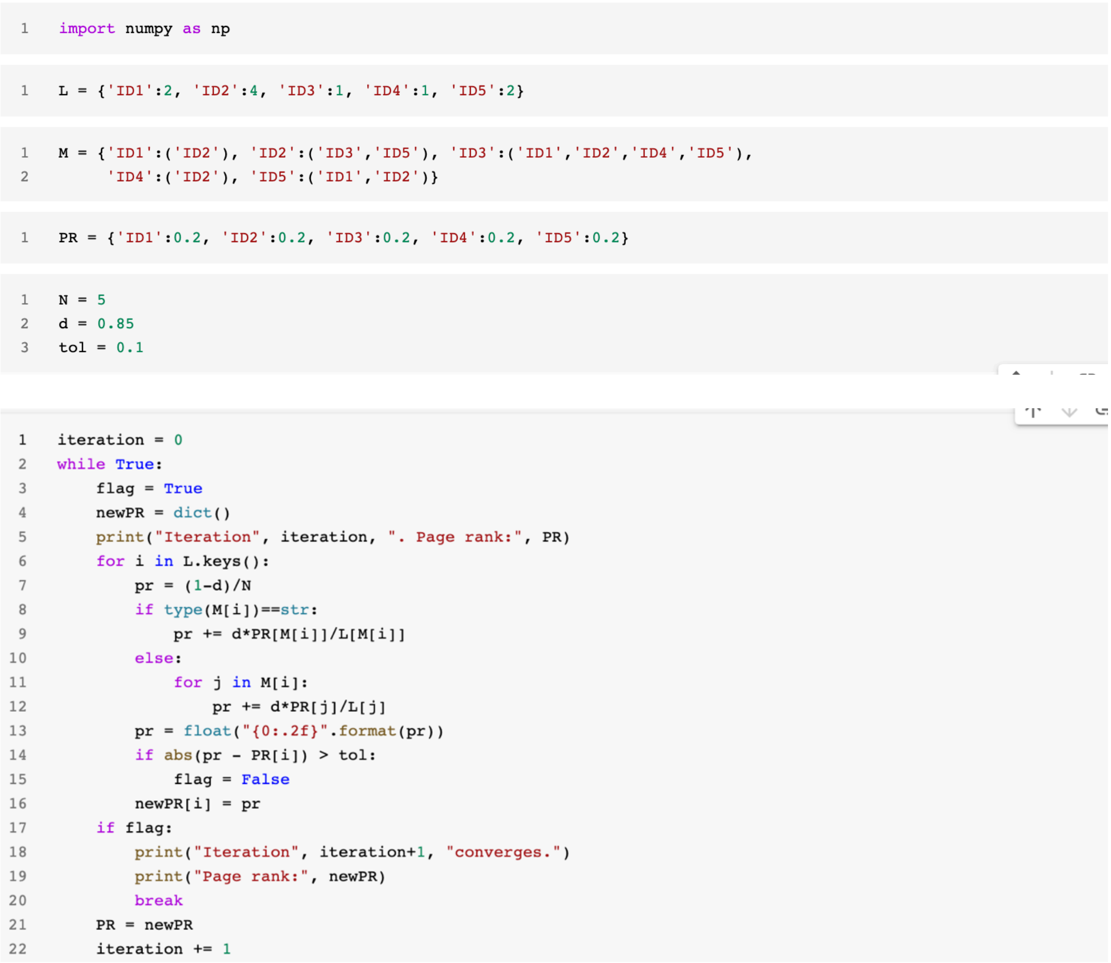
```
In [123]: result.filter("id=15496").show()

          +-----+---------+
          |   id|component|
          +-----+---------+
          |15496|        0|
          +-----+---------+

In [124]: g.edges.filter("src=15496" or "dst=15496").count()

Out[124]: 100
```

(7) PageRank Calculation

For this question, I used self-defined function to do the calculation and the result is as following along with the code.



```
Iteration 0 . Page rank: {'ID1': 0.2, 'ID2': 0.2, 'ID3': 0.2, 'ID4': 0.2, 'ID5': 0.2}
Iteration 1 . Page rank: {'ID1': 0.07, 'ID2': 0.29, 'ID3': 0.41, 'ID4': 0.07, 'ID5': 0.16}
Iteration 2 . Page rank: {'ID1': 0.09, 'ID2': 0.45, 'ID3': 0.25, 'ID4': 0.09, 'ID5': 0.12}
Iteration 3 . Page rank: {'ID1': 0.13, 'ID2': 0.29, 'ID3': 0.29, 'ID4': 0.13, 'ID5': 0.16}
Iteration 4 converges.
Page rank: {'ID1': 0.09, 'ID2': 0.34, 'ID3': 0.33, 'ID4': 0.09, 'ID5': 0.15}
```



```
import numpy as np
```

```
L = {'ID1':2, 'ID2':4, 'ID3':1, 'ID4':1, 'ID5':2}
```

```
M = {'ID1':('ID2'), 'ID2':('ID3','ID5'), 'ID3':('ID1','ID2','ID4','ID5'),
     'ID4':('ID2'), 'ID5':('ID1','ID2')}
```

```
PR = {'ID1':0.2, 'ID2':0.2, 'ID3':0.2, 'ID4':0.2, 'ID5':0.2}
```

```
N = 5
d = 0.85
tol = 0.1
```

```
iteration = 0
while True:
    flag = True
    newPR = dict()
    print("Iteration", iteration, ". Page rank:", PR)
    for i in L.keys():
        pr = (1-d)/N
        if type(M[i])==str:
            pr += d*PR[M[i]]/L[M[i]]
        else:
            for j in M[i]:
                pr += d*PR[j]/L[j]
        pr = float("{0:.2f}".format(pr))
        if abs(pr - PR[i]) > tol:
            flag = False
        newPR[i] = pr
    if flag:
        print("Iteration", iteration+1, "converges.")
        print("Page rank:", newPR)
        break
    PR = newPR
    iteration += 1
```