Data mining

In this exercise we test some graph analysis and do little analysis with the SNAP social circles data, The Facebook dataset was obtained from the Stanford web page: https://snap.stanford.edu/data/egonets-Facebook.html

Pixiedust installation

Run these code blocks once after startup or kernel restart. Fail to follow these literally will result in all kinds of mysterious failures!

Run the following command and then RESET kernel.

```
!pip install --upgrade --user pixiedust
#restart kernel now!
```

Run the following command and then RESET kernel again.

```
import pixiedust
#restart kernel now!
```

Pixiedust is now installed.

Exercise setup code

Run the following code block before each exercise. The following code should also be called after you have reset your notebook.

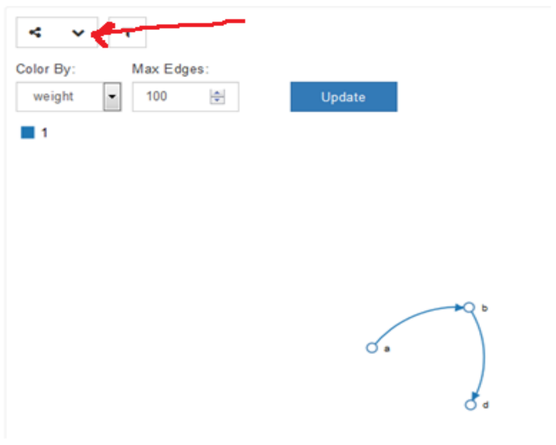
```
import os
#download graphframes package
!wget -q --show-progress http://dl.bintray.com/spark-
packages/maven/graphframes/graphframes/0.3.0-spark2.0-s_2.11/graphframes-0.3.0-
spark2.0-s_2.11.jar -P /home/jovyan/
#tell to load graphframes and dependencies to the spark cluster for use
os.environ["PYSPARK_SUBMIT_ARGS"] = ' --packages graphframes:graphframes:0.3.0-
spark2.0-s_2.11 --jars /home/jovyan/.ivy2/jars/com.typesafe.scala-logging_scala-
logging-api_2.11-2.1.2.jar,/home/jovyan/.ivy2/jars/org.scala-lang_scala-reflect-
2.11.0.jar,/home/jovyan/.ivy2/jars/com.typesafe.scala-logging_scala-logging-
slf4j_2.11-2.1.2.jar,/home/jovyan/.ivy2/jars/org.slf4j_slf4j-api-
1.7.7.jar,/home/jovyan/.ivy2/jars/graphframes_graphframes-0.3.0-spark2.0-
s_2.11.jar pyspark-shell'

import pyspark
from pyspark.sql import *

try:
    sc = pyspark.SparkContext('local[*]',environment = {})
except:
    sc = sc
#create sqlcontext on the spark, enables the use of the SQL queries below
sqlContext = SQLContext(sc)

import pixiedust     #important, pixiedust must be initialized after pyspark,
otherwise it just doesn't work right
```

1. Graphs and PageRank algorithm

Task 1: Modify code to form the graph that is shown in the *mining massive datasets* book link analysis chapter figure 5.1. Additionally, give random numbers to the edges (replace corresponding 1 in each row like (“a”,”b”,1)).

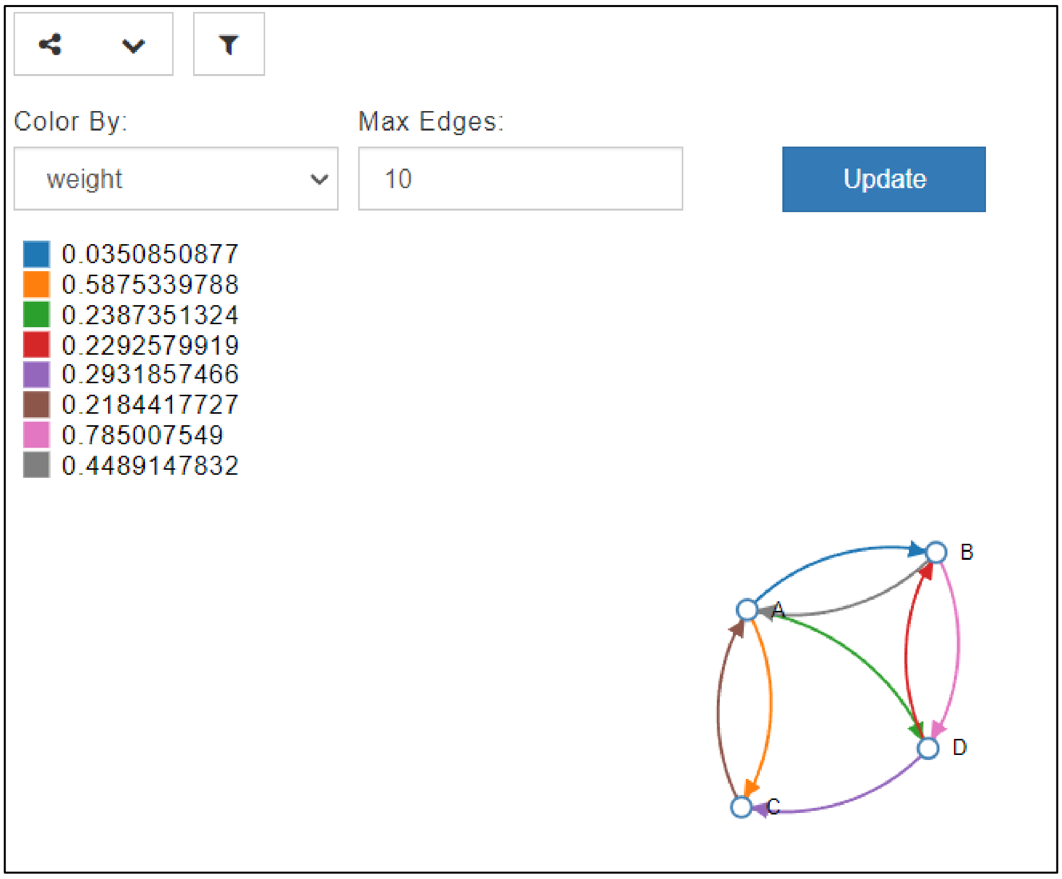
```
# Create a Vertex DataFrame with unique ID column "id"
v = sqlContext.createDataFrame([
  ("a", "Aval"),
  ("b", "Bval"),
  ("c", "Cval"),
  ("d", "Dval"),
], ["id", "value"])
# Create an Edge DataFrame with "src" and "dst" columns
e = sqlContext.createDataFrame([
  ("a", "b", 1),
  ("b", "d", 1),
], ["src", "dst", "somevalue"])
# Create a GraphFrame
import pixiedust
from graphframes import *
g = GraphFrame(v, e)

display(g)
```

Switch graph views from the menu dropdown.



Answer:

```
import random

# Create a Vertex DataFrame with unique ID column "id"
v = sqlContext.createDataFrame([
  ("A", "A"),
  ("B", "B"),
  ("C", "C"),
  ("D", "D"),
], ["id", "value"])

# Create an Edge DataFrame with "src", "dst", and "weight" columns
e = sqlContext.createDataFrame([
  ("A", "B", random.random()),
  ("A", "C", random.random()),
  ("A", "D", random.random()),
  ("D", "B", random.random()),
  ("D", "C", random.random()),
  ("C", "A", random.random()),
  ("B", "D", random.random()),
  ("B", "A", random.random()),
], ["src", "dst", "weight"])

# Create a GraphFrame
from graphframes import GraphFrame
g = GraphFrame(v, e)

display(g)
```



🟦 0.0350850877
🟧 0.5875339788
🟩 0.2387351324
🟥 0.2292579919
🟪 0.2931857466
🟫 0.2184417727
🟪 0.785007549
⬛ 0.4489147832

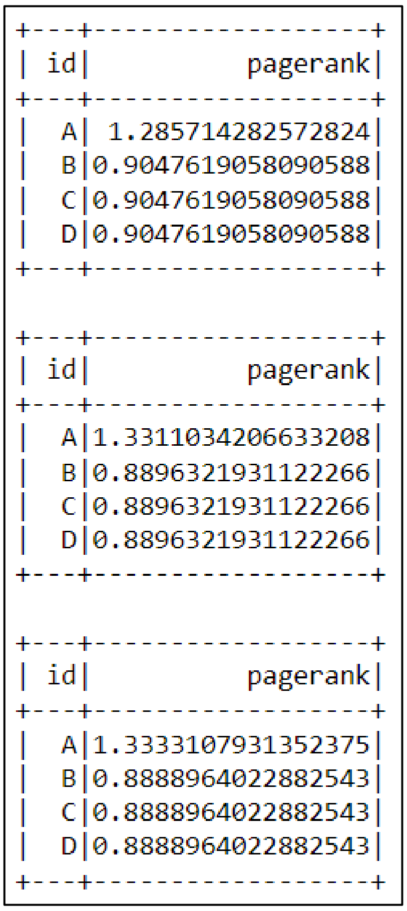
Task 2: Calculate *PageRank* (use new code block) for this matrix using *resetProbability* values of 0.2, 0.01 and 0.0001. Did the pageRank change?

```
# Run PageRank algorithm, and show results.
results = g.pageRank(resetProbability=0.01, maxIter=40)
results.vertices.select("id", "pagerank").sort("id").show()
```

Answer:

```
# Run PageRank algorithm with resetProbability = 0.2 and maxIter = 20
results_0_2 = g.pageRank(resetProbability=0.2, maxIter=20)
results_0_2.vertices.select("id", "pagerank").sort("id").show()

# Run PageRank algorithm with resetProbability = 0.01 and maxIter = 20
results_0_01 = g.pageRank(resetProbability=0.01, maxIter=20)
results_0_01.vertices.select("id", "pagerank").sort("id").show()
```

```
# Run PageRank algorithm with resetProbability = 0.0001 and maxIter = 20
results_0_0001 = g.pageRank(resetProbability=0.0001, maxIter=20)
results_0_0001.vertices.select("id", "pagerank").sort("id").show()
```



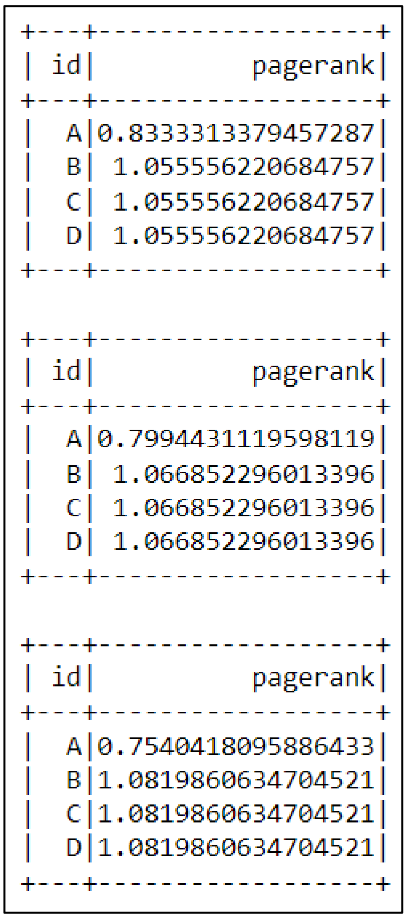
```
+---+------------------+
| id|          pagerank|
+---+------------------+
|  A|  1.28571428257282 4|
|  B|0.9047619058090588|
|  C|0.9047619058090588|
|  D|0.9047619058090588|
+---+------------------+

+---+------------------+
| id|          pagerank|
+---+------------------+
|  A|1.3311034206633208|
|  B|0.8896321931122266|
|  C|0.8896321931122266|
|  D|0.8896321931122266|
+---+------------------+

+---+------------------+
| id|          pagerank|
+---+------------------+
|  A|1.3333107931352375|
|  B|0.8888964022882543|
|  C|0.8888964022882543|
|  D|0.8888964022882543|
+---+------------------+
```

Yes, the PageRank values have changed for each reset probability value.

Task 3: Calculate *PageRank* for matrix in course book figure 5.3 using *resetProbability* values of 0.2, 0.01 and 0.0001. How did the pageRank change in this case? What is the purpose of *resetProbability* (teleportation in book) constant in the matrix equation? (See 5.1.5 Spider traps and taxation in the course book.)

Answer:

```
# Create a Vertex DataFrame with unique ID column "id"
v = sqlContext.createDataFrame([
  ("A", "value1"),
  ("B", "value2"),
  ("C", "value3"),
  ("D", "value3"),
], ["id", "value"])

# Create an Edge DataFrame with "src" and "dst" columns
e = sqlContext.createDataFrame([
  ("A", "B", random.random()),
  ("A", "C", random.random()),
  ("A", "D", random.random()),
  ("D", "B", random.random()),
  ("D", "C", random.random()),
  ("B", "D", random.random()),
  ("B", "A", random.random()),
], ["src", "dst", "weight"])

# Create a GraphFrame
from graphframes import GraphFrame
g = GraphFrame(v, e)

# Calculate PageRank with resetProbability = 0.2 and maxIter = 20
results_0_2 = g.pageRank(resetProbability=0.2, maxIter=20)
results_0_2.vertices.select("id", "pagerank").sort("id").show()

# Calculate PageRank with resetProbability = 0.01 and maxIter = 20
results_0_01 = g.pageRank(resetProbability=0.01, maxIter=20)
results_0_01.vertices.select("id", "pagerank").sort("id").show()

# Calculate PageRank with resetProbability = 0.0001 and maxIter = 20
results_0_0001 = g.pageRank(resetProbability=0.0001, maxIter=20)
results_0_0001.vertices.select("id", "pagerank").sort("id").show()
```



```
+---+------------------+
| id|          pagerank|
+---+------------------+
|  A|0.8333313379457287|
|  B| 1.055556220684757|
|  C| 1.055556220684757|
|  D| 1.055556220684757|
+---+------------------+

+---+------------------+
| id|          pagerank|
+---+------------------+
|  A|0.7994431119598119|
|  B| 1.066852296013396|
|  C| 1.066852296013396|
|  D| 1.066852296013396|
+---+------------------+

+---+------------------+
| id|          pagerank|
+---+------------------+
|  A|0.7540418095886433|
|  B|1.0819860634704521|
|  C|1.0819860634704521|
|  D|1.0819860634704521|
+---+------------------+
```

The PageRank values changed for different reset probability values. Lower reset probabilities distribute importance more evenly among the nodes, resulting in relatively higher PageRank values for some nodes. Higher reset probabilities concentrate importance on the initial node, leading to a higher PageRank value for that node. The resetProbability constant (teleportation probability) in the matrix equation introduces randomness, preventing the algorithm from getting stuck and improving the robustness and convergence of PageRank. Its purpose is to ensure the algorithm explores the entire graph and handles scenarios like spider traps, providing more accurate and fair rankings.

2. Social network graph (remember setup code from start)

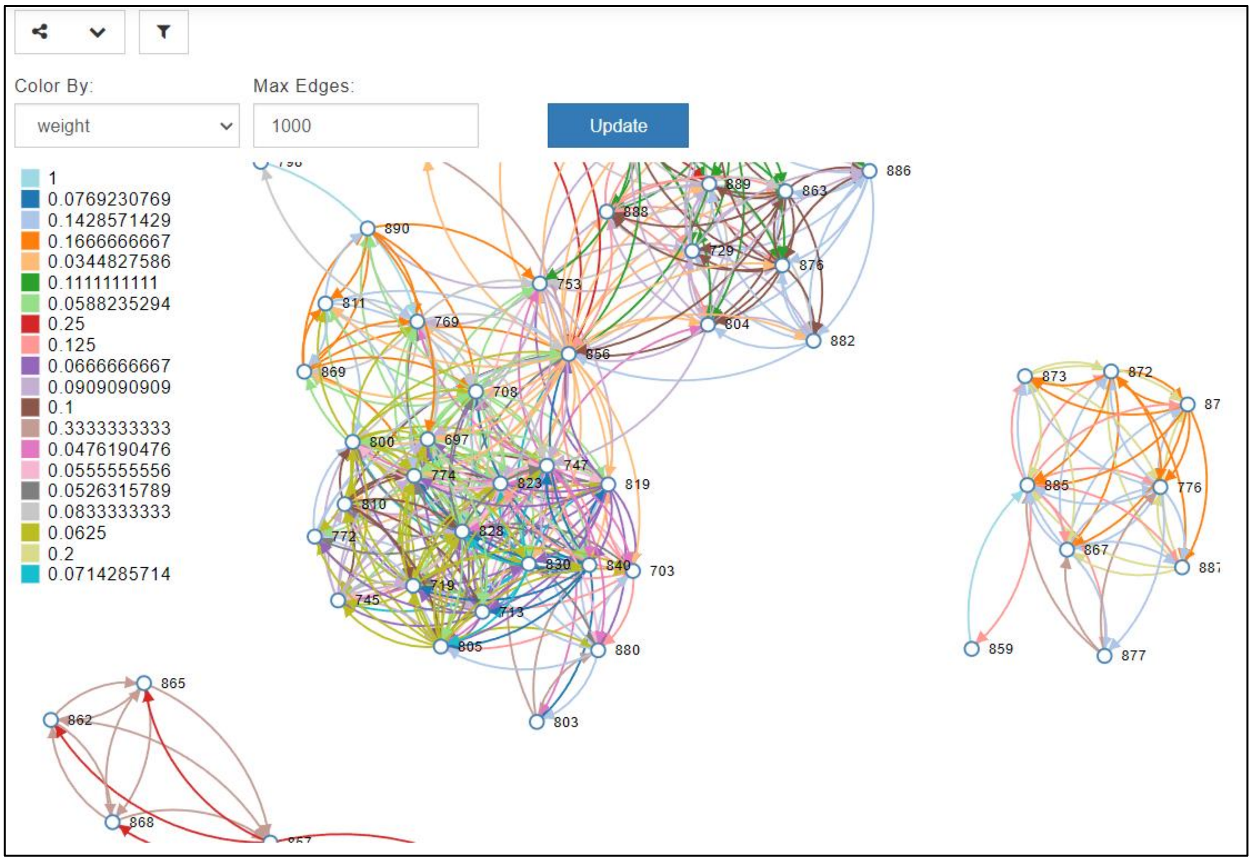
Upload and unzip e4_data.zip file in console. Or unzip files locally and upload files to notebook.

Task 1: Run the code below and try to inspect the graph and the table, if the key users (graph vertices) could be found using the PageRank algorithm. Select *bidirect graph* from *pixiedust* plot menu and set *max edges* to 1000 and press *update*. You can drag nodes to see slightly better how they are connected.

Answer:

```
#read graph edges(or arcs, lines) (multiple edges per vertex can exist)
lines = sc.textFile("698.edges")
edges = lines.map(lambda l: l.split(" ")). \
            map(lambda p: Row( src=int(p[0]), dst=int(p[1])) )
edges = sqlContext.createDataFrame(edges)

#read graph vertices(or nodes, points) (these are unique)
lines = sc.textFile("698.feat")
vertices = lines.map(lambda l: l.split(" ")). \
            map(lambda p: Row(id=int(p[0]), name="userid_"+p[0]) )
vertices = sqlContext.createDataFrame(vertices)

# Create a GraphFrame
from graphframes import *
g = GraphFrame(vertices, edges)

#calculate pagerank
pagerank = g.pageRank(resetProbability=0.01, maxIter=20)
degrees = g.degrees

#join two results
result = pagerank.vertices.join(degrees,"id")
#print
result.select("name","pagerank","degree").orderBy("pagerank",
ascending=False).show(100,False)

display(pagerank)
```

```
+----------+------------------+------+
|name      |pagerank          |degree|
+----------+------------------+------+
|userid_856|2.869904930044944 |58    |
|userid_747|2.0262853666897436|42    |
|userid_828|1.8265794370043782|38    |
|userid_823|1.7310091858175893|36    |
|userid_708|1.6420290157745947|34    |
|userid_697|1.6384157371799375|34    |
|userid_774|1.6375792879752207|34    |
|userid_885|1.624070364414393 |16    |
|userid_800|1.5407748992673922|32    |
|userid_805|1.536157497843886 |32    |
|userid_719|1.5355823021086208|32    |
|userid_871|1.4709393973810154|12    |
|userid_870|1.449453946107641 |12    |
|userid_881|1.4488520165515446|12    |
|userid_819|1.445161790912115 |30    |
|userid_713|1.4396872605542013|30    |
|userid_776|1.4152246593339932|14    |
|userid_867|1.4152246593339932|14    |
|userid_830|1.3461881979353485 |28    |
|userid_840|1.2537690367314105 |26    |
|userid_889|1.2215494979864316 |24    |
|userid_872|1.2132072804684861 |12    |
|userid_874|1.2132072804684861 |12    |
|userid_895|1.2093909791788997 |10    |
|userid_879|1.2093816448687078 |10    |
|userid_769|1.1765777187175621 |24    |
|userid_729|1.115746927091516  |22    |
|userid_804|1.1139446959561388 |22    |
|userid_753|1.0821699270402492 |22    |
|userid_857|1.0665987460168191 |8     |
|userid_745|1.0574079004034842 |22    |
|userid_863|1.0179571460976646 |20    |
|userid_876|1.0179571460976644 |20    |
|userid_873|1.0124546847076596 |10    |
|userid_887|1.0124546847076596 |10    |
|userid_858|0.9698390523257981 |8     |
|userid_866|0.9672665010765201 |8     |
|userid_810|0.9646941304483965 |20    |
|userid_878|0.9165600287607797 |18    |
|userid_861|0.912671314354855  |18    |
|userid_862|0.8178867126752429 |6     |
|userid_865|0.8178867126752429 |6     |
|userid_868|0.8178867126752429 |6     |
|userid_888|0.8148011153129138 |16    |
|userid_703|0.7756857750338367 |16    |
|userid_894|0.727590551440297  |6     |
|userid_886|0.7174943521450154 |14    |
|userid_882|0.7148543474559972 |14    |
|userid_811|0.6866737027190494 |14    |
|userid_880|0.6806413989853524 |14    |
|userid_772|0.6791718260381926 |14    |
|userid_877|0.6120963512452997 |6     |
|userid_869|0.5910333006556802 |12    |
|userid_890|0.5905450295675715 |12    |
|userid_864|0.4333621472648841 |8     |
|userid_825|0.427687503552187  |8     |
|userid_884|0.33271893040404016|6     |
|userid_893|0.3266584286128287 |6     |
|userid_803|0.2977556405450429 |6     |
|userid_859|0.21178976504976002|2     |
|userid_798|0.1077993661565252 |2     |
+----------+-------------------+------+
```


Color By: weight

Max Edges: 1000

Task 2: Is the PageRank algorithm any better than just counting edges (graph degree) to find the most important vertices of the network? Give some examples which user ids you think are important by visual inspection, and in pagerank or degree. *(For some analyses a vertex can be important if it connects network parts that would be isolated without the connecting vertex.)*

Answer:

The PageRank algorithm is more effective than simply counting edges (graph degree) to identify important vertices in a network. PageRank considers both the number and quality of connections, while degree only measures the number of connections.

In the provided code, PageRank is calculated using g.pageRank(), and the graph degree is obtained using g.degrees. By comparing the results, we can identify important vertices based on PageRank and degree.

To identify important vertices visually, consider the following:

- High PageRank: These vertices are frequently visited or referenced in the network.
- High degree: These vertices have many connections to other vertices, facilitating overall network connectivity.

To see the results, check the output of result.select("name", "pagerank", "degree").orderBy("pagerank", ascending=False).show(100, False). This displays the names, PageRank scores, and degrees of the top vertices in descending order of PageRank.

```
#read graph edges(or arcs, lines) (multiple edges per vertex can exist)
lines = sc.textFile("698.edges")
edges = lines.map(lambda l: l.split(" ")). \
        map(lambda p: Row( src=int(p[0]), dst=int(p[1])) )
edges = sqlContext.createDataFrame(edges)

#read graph vertices(or nodes, points) (these are unique)
```

```
lines = sc.textFile("698.feat")
vertices = lines.map(lambda l: l.split(" ")). \
        map(lambda p: Row(id=int(p[0]), name="userid_"+p[0]) )
vertices = sqlContext.createDataFrame(vertices)

# Create a GraphFrame
from graphframes import *
g = GraphFrame(vertices, edges)

#calculate pagerank
pagerank = g.pageRank(resetProbability=0.01, maxIter=20)
degrees = g.degrees

#join two results
result = pagerank.vertices.join(degrees,"id")
#print
result.select("name","pagerank","degree").orderBy("pagerank", ascending=False).show(100,False)

display(pagerank)
```

3. Analyzing social network clustering (remember setup)

A graph can be split into clusters by the connectivity, contents or with both connectivity and contents of the graph. In social network graphs individuals may belong into many groups or communities so the clustering is not strictly defined for this kind of data.

A graph can be partitioned in many ways and the graph partition problem is NP hard to find the best partition. A big computation cluster seems to be the only way to find good solution for large graphs quickly. As seen in the course book chapter Mining Social-Network Graphs, there is many different graph clustering/partitioning/grouping algorithms available.

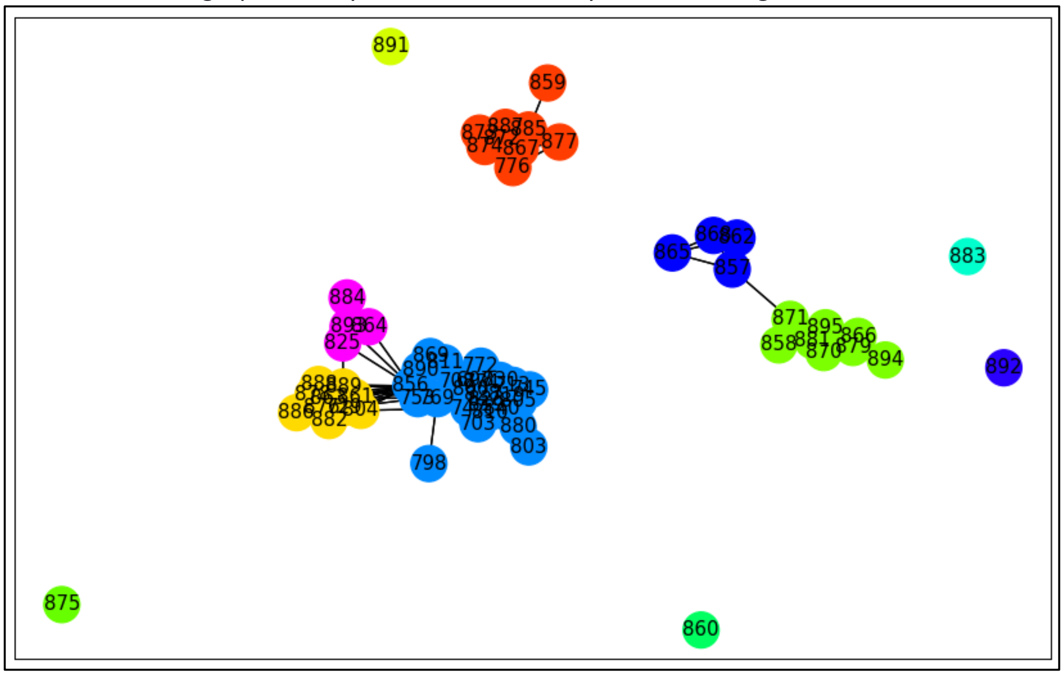
In the following code, Label Propagation Algorithm is used to cluster communities. Start with the uploaded data files from the last example.

Sadly, we cannot easily plot this with pixiedust, so I had to revert to old deprecated plot. Please ignore warnings. Maybe things get improved next year.

```
%matplotlib notebook

import networkx as nx
import matplotlib.pyplot as plt
import numpy as nb

#readn graph edges(or arcs, lines) (multiple edges per vertex can exist)
lines = sc.textFile("698.edges")
edges = lines.map(lambda l: l.split(" ")). \
            map(lambda p: Row( src=int(p[0]), dst=int(p[1])) )
edges = sqlContext.createDataFrame(edges)

#read graph vertices(or nodes, points) (these are unique)
lines = sc.textFile("698.feat")
vertices = lines.map(lambda l: l.split(" ")). \
            map(lambda p: Row(id=int(p[0]), name="userid_"+p[0]
,features=[int(x) for x in p[1:]] ))
vertices = sqlContext.createDataFrame(vertices)

# Create a GraphFrame
from graphframes import *
g = GraphFrame(vertices, edges)

#calculate simple clustering with the label propagation clustering alg.
lpa = g.labelPropagation(maxIter=5)
nodes = lpa.select("id","label")

#plotting, generate unique colors for each group-----------------------
G = nx.DiGraph()
for x in g.edges.collect():
    G.add_edges_from([(x[0],x[1])], weight=1)
for x in lpa.select("id","label").rdd.map(lambda r: ( int(r[0]),int(r[1]))
).collect():
    G.add_node(x[0],label=x[1])
grouplabels = [list(x[1].values())[0] for x in G.nodes(True)]
node_texts = {node:node for node in G.nodes()};
cmap = plt.get_cmap('gist_rainbow')
uniqlabels = nb.unique(grouplabels)
randvals = nb.random.random_sample((len(uniqlabels),1))
colorlut = dict(zip(uniqlabels,randvals))
gcolors = []
```

```
for x in grouplabels:
    gcolors.append(cmap(float(colorlut[x])))
positions=nx.spring_layout(G,k=0.1,scale=1.5,iterations=20)
nx.draw_networkx(G,positions, labels=node_texts, node_color = gcolors,
node_size=500,arrows=False)
#plotting end -----------------------------------------------------------
```

Task 1: Examine graph and try to count how many clusters LPA generated.



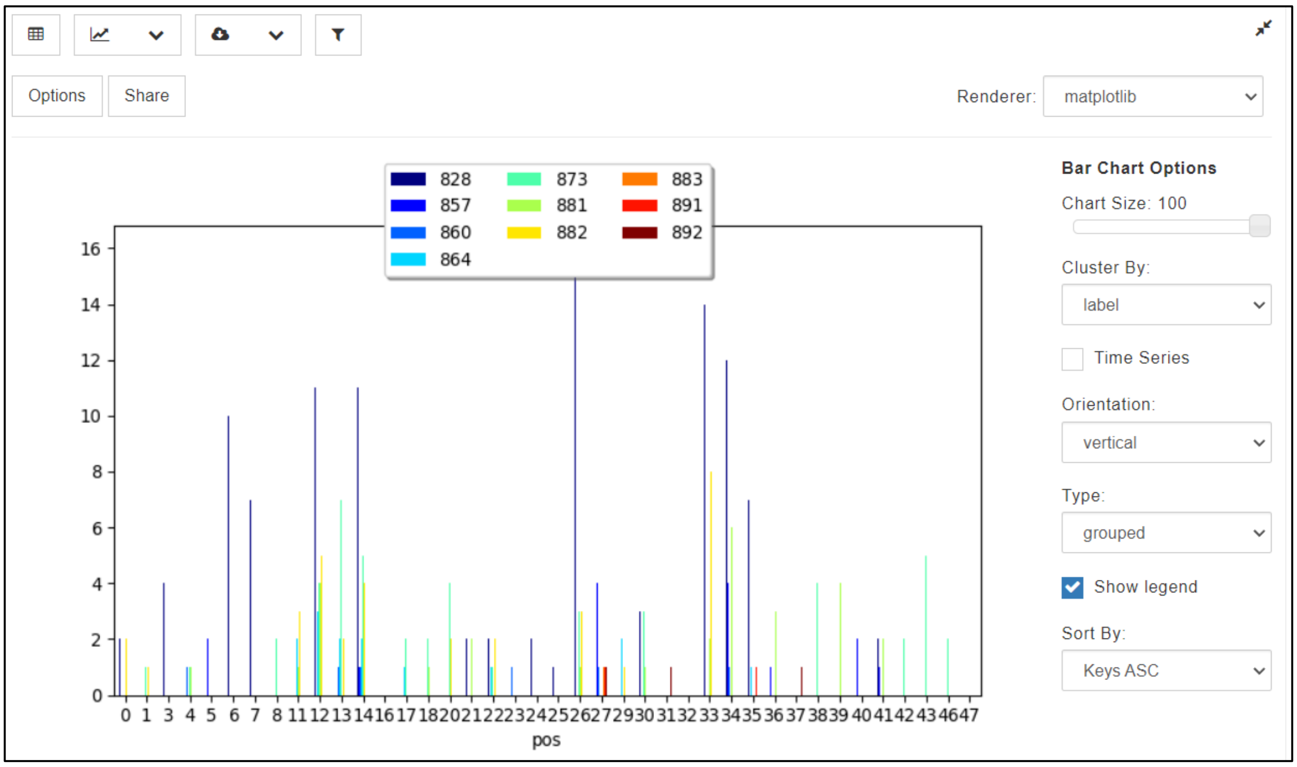
Answer:

There are a few clusters (more than eight) that LPA has generated.

In the next phase we analyze if our individuals in our clusters have some features that are common for them.

Task 2: Add and run following code. Try to make histogram plot with pixiedust (try bar chart with count aggrecation) pos = x-axis, col=y-axis. Finally show all labeled clusters by selecting *"Cluster by" label*.

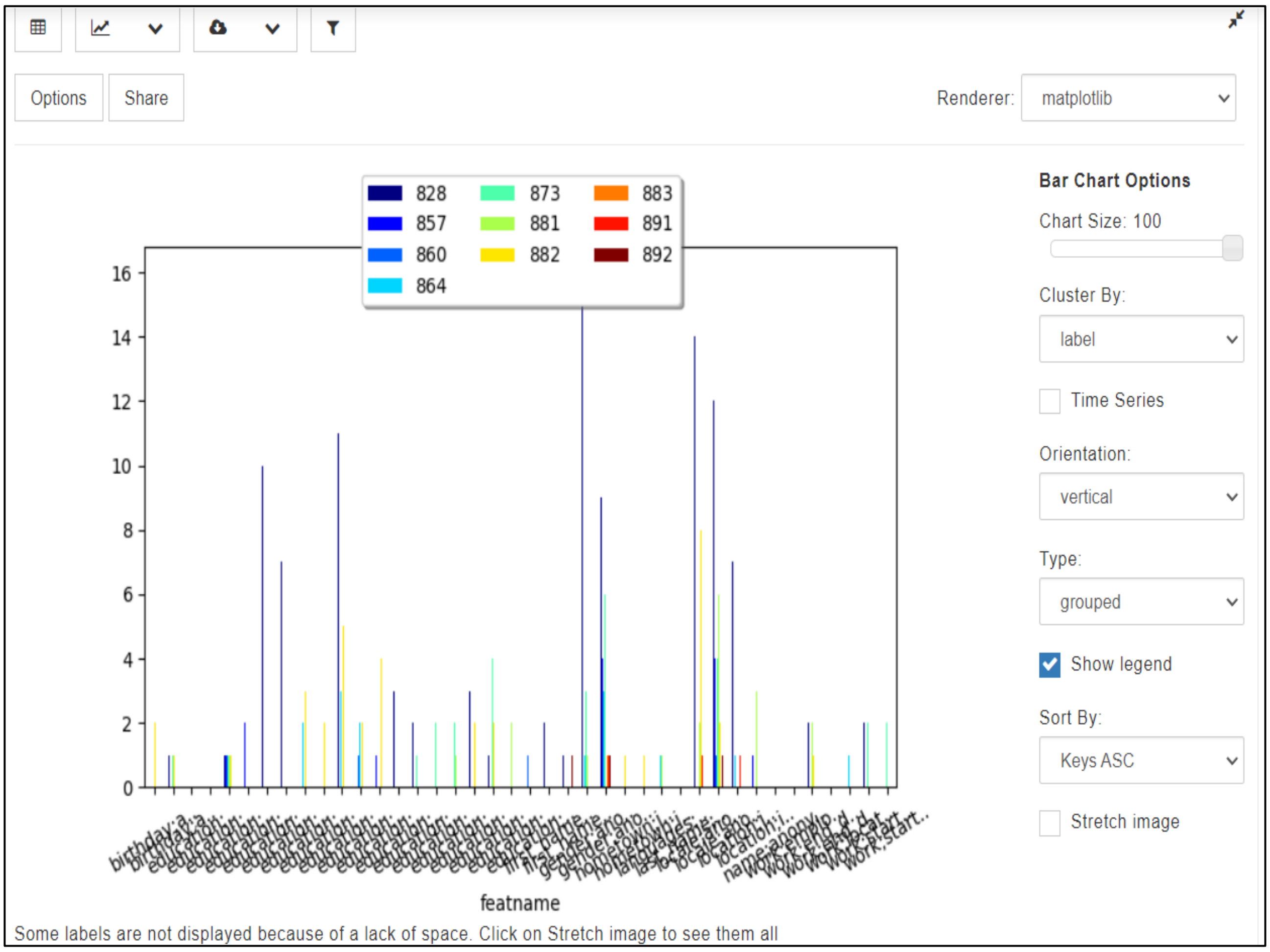
```
#line code explodes features into key values and duplicates label and ids
from pyspark.sql.functions import posexplode
#remove zeros from features to make histogram plotting easier
res = lpa.select('id','label',posexplode(lpa.features)).filter("col>0")
display(res)
```



Task 3: Import feature names from the corresponding *.featnames* ending filename. Give the id number column name *"pos" and feature name "featname"* to ease join operation. Please also concatenate the last number from each featurename line to the end of featname. Do a join into "res" *dataFrame* with *"pos"* column. Finally modify bar graph to display feature names on histogram x-axis.

Answer:

```
from pyspark.sql.functions import posexplode

# Explode features into key values and duplicate label and ids
res = lpa.select('id', 'label', posexplode(lpa.features)).filter("col > 0")

# Import feature names from the corresponding .featnames file
featnames_lines = sc.textFile("698.featnames")
featnames = featnames_lines.map(lambda l: l.split(" ")).map(lambda p:
Row(pos=int(p[0]), featname=" ".join(p[1:]) + " " + p[-1]))
featnames_df = sqlContext.createDataFrame(featnames)

# Join the "res" DataFrame with the "featnames" DataFrame based on the "pos" column
joined_res = res.join(featnames_df, "pos")

# Modify bar graph to display feature names on histogram x-axis
import pixiedust
display(joined_res)
```



Renderer: matplotlib

Bar Chart Options

Chart Size: 100

Cluster By: label

Time Series [ ]

Orientation: vertical

Type: grouped

Show legend [✓]

Sort By: Keys ASC

Stretch image [ ]

Some labels are not displayed because of a lack of space. Click on Stretch image to see them all

Task 4: Do you see any distinguishing patterns in the pixiedust plots? Do all label groups form meaningful histograms for analysis (enough data)? You can expand the names by ticking *"stretch image"* checkbox.

Answer:

The histograms we made using pixiedust show different patterns that help us see differences between the groups of labels. Some groups have clear peaks or clusters in their histograms, which means that the individuals in those groups have similar features. However, not all groups have enough data or noticeable differences in features, so their histograms may not give us much useful information. When we expand the feature names, we can better understand how important those features are in explaining the patterns we see. It's important to think about these findings in the context of the specific data and what we already know about the subject.

Task 5: Describe what the following code prints out. Requires task 3 to be completed!

```
uniqvals = res.select('label').distinct().rdd.map(lambda p: p[0]).collect()
res.persist() #looping queries following, store intermediate data
for x in uniqvals:
   print("Group: " + str(x) + "\n")
   ures = res.filter('label=' +
str(x)).groupBy("featname").sum("col").sort("sum(col)",ascending=False).rdd.map(lambda p:
(p[0],p[1])).take(10)
   print(ures)
   print('\n')
```

Answer:

The code prints the results of a data analysis for each unique label in the 'res' DataFrame. It filters the DataFrame based on the label, groups the data by feature name, calculates the sum of a column, and sorts the groups.

4. Visualize flight routes data as graph (remember setup) (optional)

Task 1: With the *routes.dat* and *airports.dat* already introduces in the exercise document 2, construct a graph that visualizes the flight route network. You can also reuse parts of the exercise code to import dataFrames! Remember to set source and destination names (like 'LAX') to *src* and *dst* columns. While importing *airports.dat* set *airport name* to id column (id column should also have three letter ids' like 'LAX').
Run *display* command on the graph now and select max vertices as 1000. Now you may notice that your plot result is total nonsense, because the plot cannot handle all vertices and edges. Too much data to view is a common problem in data analysis and a big part of data mining is all about how to condense your result small enough to be understandable for target audience.

Answer:

```
from graphframes import GraphFrame
from pyspark.sql import SparkSession, Row

# Create SparkSession
spark = SparkSession.builder.appName("Flight Route Network").getOrCreate()

# Read airports.dat file and create vertices DataFrame
airports_lines = spark.sparkContext.textFile("airports.dat")
airports_data = airports_lines.map(lambda l: l.split(",")).map(lambda p: Row(id=p[4], name=p[1]))
vertices_df = spark.createDataFrame(airports_data)

# Read routes.dat file and create edges DataFrame
routes_lines = spark.sparkContext.textFile("routes.dat")
routes_data = routes_lines.map(lambda l: l.split(",")).map(lambda p: Row(src=p[2], dst=p[4]))
edges_df = spark.createDataFrame(routes_data)

# Create GraphFrame
g = GraphFrame(vertices_df, edges_df)

# Visualize the graph
g.vertices.show()
g.edges.show()

# Stop SparkSession
spark.stop()
```

```
+-----+--------------------+
|   id|                name|
+-----+--------------------+
|"GKA"|    "Goroka Airport"|
|"MAG"|    "Madang Airport"|
|"HGU"|"Mount Hagen Kaga...|
|"LAE"|    "Nadzab Airport"|
|"POM"|"Port Moresby Jac...|
|"WWK"|"Wewak Internatio...|
|"UAK"|"Narsarsuaq Airport"|
|"GOH"|"Godthaab / Nuuk ...|
|"SFJ"|"Kangerlussuaq Ai...|
|"THU"|    "Thule Air Base"|
|"AEY"|  "Akureyri Airport"|
|"EGS"|"Egilsstaðir Airp...|
|"HFN"|"Hornafjörður Air...|
|"HZK"|   "Húsavík Airport"|
|"IFJ"|"Ísafjörður Airport"|
|"KEF"|"Keflavik Interna...|
|"PFJ"|"Patreksfjörður A...|
|"RKV"| "Reykjavik Airport"|
|"SIJ"|"Siglufjörður Air...|
|"VEY"|"Vestmannaeyjar A...|
+-----+--------------------+
only showing top 20 rows

+---+---+
|src|dst|
+---+---+
|AER|KZN|
|ASF|KZN|
|ASF|MRV|
|CEK|KZN|
|CEK|OVB|
|DME|KZN|
|DME|NBC|
|DME|TGK|
|DME|UUA|
|EGO|KGD|
|EGO|KZN|
|GYD|NBC|
|KGD|EGO|
|KZN|AER|
|KZN|ASF|
|KZN|CEK|
|KZN|DME|
|KZN|EGO|
|KZN|LED|
|KZN|SVX|
+---+---+
only showing top 20 rows
```

Task 2: Add Airline id to the edges while importing data and then find a way to filter edges in a way that only one airline is present on the graph. Airline ids are on the first column in the *routes.dat*. Try at first the "2B" airline.

Answer:

```
from graphframes import GraphFrame
from pyspark.sql import SparkSession, Row

# Create SparkSession
spark = SparkSession.builder.appName("Flight Route Network").getOrCreate()

# Read airports.dat file and create vertices DataFrame
airports_lines = spark.sparkContext.textFile("airports.dat")
airports_data = airports_lines.map(lambda l: l.split(",")).map(lambda p: Row(id=p[4], 
name=p[1]))
vertices_df = spark.createDataFrame(airports_data)

# Read routes.dat file and create edges DataFrame with airline ID
routes_lines = spark.sparkContext.textFile("routes.dat")
routes_data = routes_lines.map(lambda l: l.split(",")).map(lambda p: Row(src=p[2], 
dst=p[4], airline_id=p[0]))
edges_df = spark.createDataFrame(routes_data)

# Filter edges DataFrame to include only the desired airline (e.g., '2B')
```

```
filtered_edges_df = edges_df.filter(edges_df.airline_id == '2B')

# Create GraphFrame
g = GraphFrame(vertices_df, filtered_edges_df)

# Visualize the graph
g.vertices.show()
g.edges.show()

# Stop SparkSession
spark.stop()
```

```
+-----+--------------------+
|   id|                name|
+-----+--------------------+
|"GKA"|    "Goroka Airport"|
|"MAG"|    "Madang Airport"|
|"HGU"|"Mount Hagen Kaga...|
|"LAE"|    "Nadzab Airport"|
|"POM"|"Port Moresby Jac...|
|"WWK"|"Wewak Internatio...|
|"UAK"|"Narsarsuaq Airport"|
|"GOH"|"Godthaab / Nuuk ...|
|"SFJ"|"Kangerlussuaq Ai...|
|"THU"|    "Thule Air Base"|
|"AEY"|  "Akureyri Airport"|
|"EGS"|"Egilsstaðir Airp...|
|"HFN"|"Hornafjörður Air...|
|"HZK"|   "Húsavík Airport"|
|"IFJ"|"Ísafjörður Airport"|
|"KEF"|"Keflavik Interna...|
|"PFJ"|"Patreksfjörður A...|
|"RKV"| "Reykjavik Airport"|
|"SIJ"|"Siglufjörður Air...|
|"VEY"|"Vestmannaeyjar A...|
+-----+--------------------+
only showing top 20 rows

+---+---+----------+
|src|dst|airline_id|
+---+---+----------+
|AER|KZN|        2B|
|ASF|KZN|        2B|
|ASF|MRV|        2B|
|CEK|KZN|        2B|
|CEK|OVB|        2B|
|DME|KZN|        2B|
|DME|NBC|        2B|
|DME|TGK|        2B|
|DME|UUA|        2B|
|EGO|KGD|        2B|
|EGO|KZN|        2B|
|GYD|NBC|        2B|
|KGD|EGO|        2B|
|KZN|AER|        2B|
|KZN|ASF|        2B|
|KZN|CEK|        2B|
|KZN|DME|        2B|
|KZN|EGO|        2B|
|KZN|LED|        2B|
|KZN|SVX|        2B|
+---+---+----------+
only showing top 20 rows
```