Import the needed packages

Entrée [1]:

```
import pandas as pd
import networkx as nt
import collections
import matplotlib.pyplot as plt
import numpy as np
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from time import time
from sklearn import metrics
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
```

Import the data of the graph

Entrée [2]:

```
data_user_user = pd.read_csv("./data/Social_spammers_dataset/graphs/c_combined_edge
data_user_app = pd.read_csv("./data/Social_spammers_dataset/graphs/app_based_simila
labels = pd.read_csv("./data/Social_spammers_dataset/users/coded_ids_labels_train.c
```

Let's see th overview of the graph data

Entrée [3]:

```
data_user_app.head()
```

Out[3]:

	user_id	app_id	weight
0	1	23	9
1	1	33	391
2	362	200	192
3	362	23	8
4	488	176	61

Entrée [4]:

```
data_user_user.head()
```

Out[4]:

	Source	Target	Weight	Sim
0	1	168	30	0.975
1	4	56	13	0.974
2	149	244	12	1.000
3	198	244	4	1.000
4	1	244	16	1.000

Get label in dictionnary to facilitate the parsing

Entrée [5]:

```
labels.head()
labels_dict=labels.to_dict()
```

Creation the graph from dataframe

With networkx lib, we will build our graph. We will build two graph. The first graph is a graph on the relationship between user. The second one is about user and application used.

Entrée [6]:

```
graph_user_app = nt.from_pandas_edgelist(data_user_app,source="user_id",target="app
graph_user_user = nt.from_pandas_edgelist(data_user_user,source="Source",target="Ta
```

The metric of graph

In this section we will do the graph exploration by computing some metrics

Entrée [7]:

```
print("Info graph user_app: ")
print(nt.info(graph_user_app))
print("Info graph user_app: ")
print(nt.info(graph_user_user))
```

```
Info graph user_app:
Name:
Type: Graph
Number of nodes: 767
Number of edges: 2021
Average degree:   5.2699
Info graph user_app:
Name:
Type: Graph
Number of nodes: 295
Number of edges: 1335
Average degree:   9.0508
```

Computing the diameter and radius of the graph

Entrée [8]:

```
print(f"radius of graph user app: {nt.radius(graph_user_app)}")
print(f"diameter of graph user app: {nt.diameter(graph_user_app)}")
```

```
radius of graph user app: 4
diameter of graph user app: 6
```

Computing of the density of graph

Entrée [9]:

```
density_user_user = nt.density(graph_user_user)
density_user_app = nt.density(graph_user_app)
print("The density of user_user_graph is {}".format(density_user_user))
print("The density of user_app_grap is {}".format(density_user_app))
```

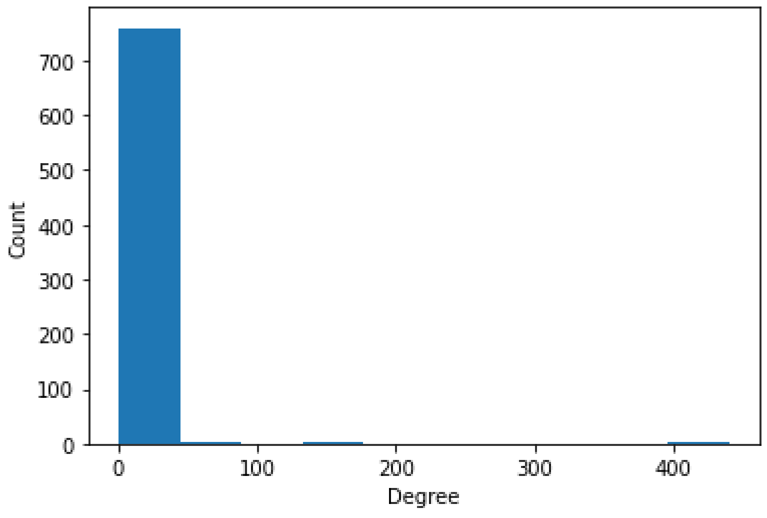
```
The density of user_user_graph is 0.030785195434105846
The density of user_app_grap is 0.006879742375604658
```

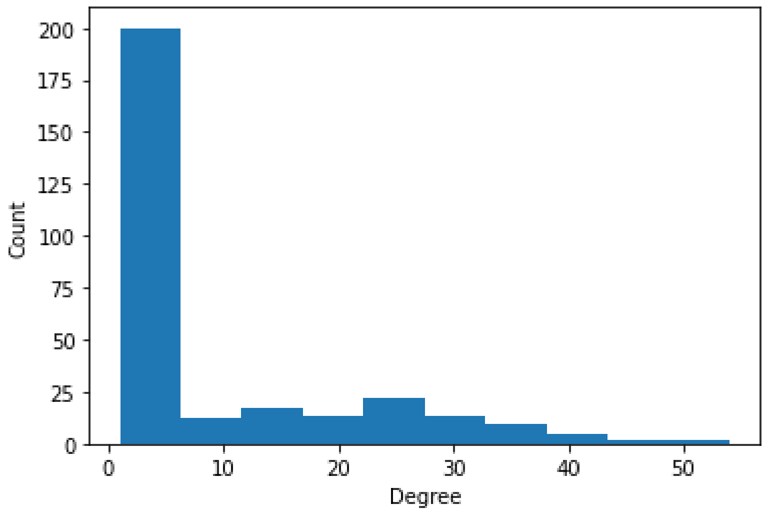
Degree ditribition analysis

Let'draw the histogram of degree distribution of our two graph

Entrée [10]:

```
def plot_degree_dist(G,name):
    fig1 = plt.gcf()
    degrees = [G.degree(n) for n in G.nodes()]
    plt.hist(degrees)
    plt.ylabel("Count")
    plt.xlabel("Degree")
    plt.show()
    #plt.draw()
    fig1.savefig(name, dpi=100)

plot_degree_dist(graph_user_app, "graph_user_app_deg_dist.png")
plot_degree_dist(graph_user_user, "graph_user_user_deg_dist.png")
```





Compute the distribution of degree only on one class

First, we will separate the node in two sets: one for the spammers and another one for the non spammer And after that we will draw the degree distribution for each class of node.

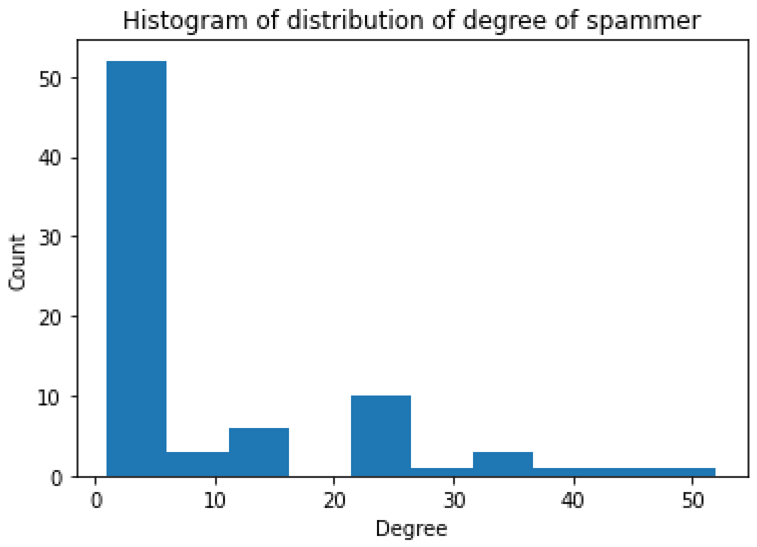
In this section we also compute the mean of the degree on each class of node

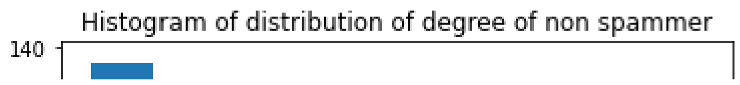
Entrée [11]:

```
#In this section we look for the spammer node and the non spammer node
list_spam = []
list_non_spam = []
for node in graph_user_user.nodes():
    if labels_dict['label'].get(node) is not None:
        if labels_dict['label'].get(node) ==1:
            list_spam.append(node)

for node in graph_user_user.nodes():
    if labels_dict['label'].get(node) is not None:
       if labels_dict['label'].get(node) ==0:
            list_non_spam.append(node)
```

Entrée [12]:

```
import numpy as np
def plot_degree_dist(G, list_node, title, name_file):
    fig1 = plt.gcf()
    degrees = [G.degree(n) for n in list_node]
    plt.hist(degrees)
    plt.ylabel("Count")
    plt.xlabel("Degree")
    plt.title(title)
    plt.show()
    fig1.savefig(name_file, dpi=100)

plot_degree_dist(graph_user_user,list_spam,"Histogram of distribution of degree of
plot_degree_dist(graph_user_user,list_non_spam,"Histogram of distribution of degree


def compute_mean_of_degree(G, list_node):
    degrees = [G.degree(n) for n in list_node]
    return np.mean(degrees)

print ("The mean of degree of the spammer: {}".format(compute_mean_of_degree(graph_
print ("The mean of degree of the non spammer: {}".format(compute_mean_of_degree(gr
```





We can constate that the mean of the degree of group of spammer is greater than the non spammer mean of degree This observation allows us to say that the spammer share more edge than the non spammer.

Next, we will get the array of the degree for each class.

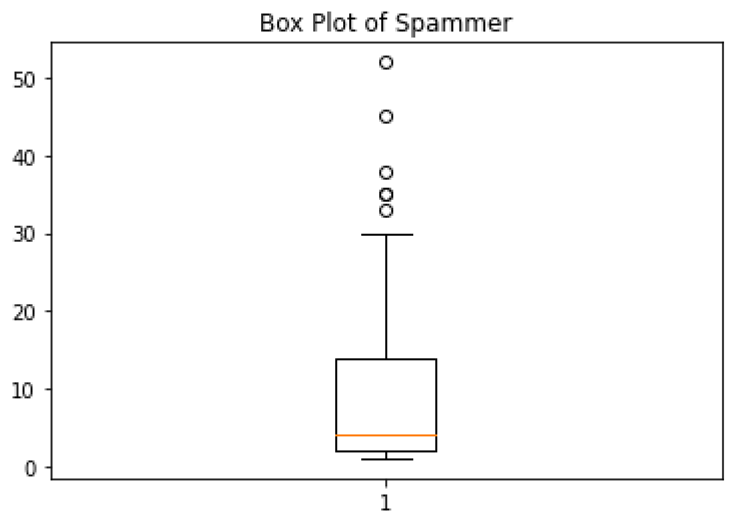
With these array, we will draw boxplot to see how it looks like.

Entrée [13]:

```
degrees_spammer = [graph_user_user.degree(n) for n in list_spam]
degrees_non_spammer = [graph_user_user.degree(n) for n in list_non_spam]
```

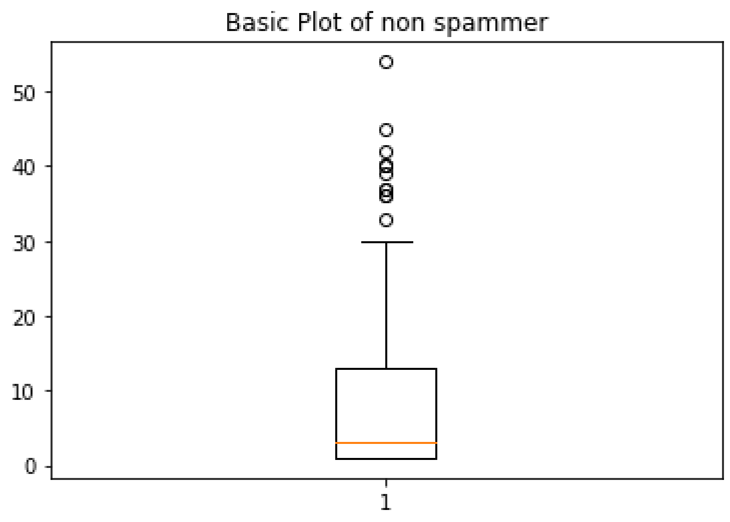
Entrée [14]:

```
fig1, ax1 = plt.subplots()
ax1.set_title('Box Plot of Spammer')
ax1.boxplot(degrees_spammer)
fig1.savefig("boxplot_spam.png", dpi=100)
```



Entrée [15]:

```
fig1, ax1 = plt.subplots()
ax1.set_title('Basic Plot of non spammer')
ax1.boxplot(degrees_non_spammer)
fig1.savefig("boxplot_non_spam.png", dpi=100)
```

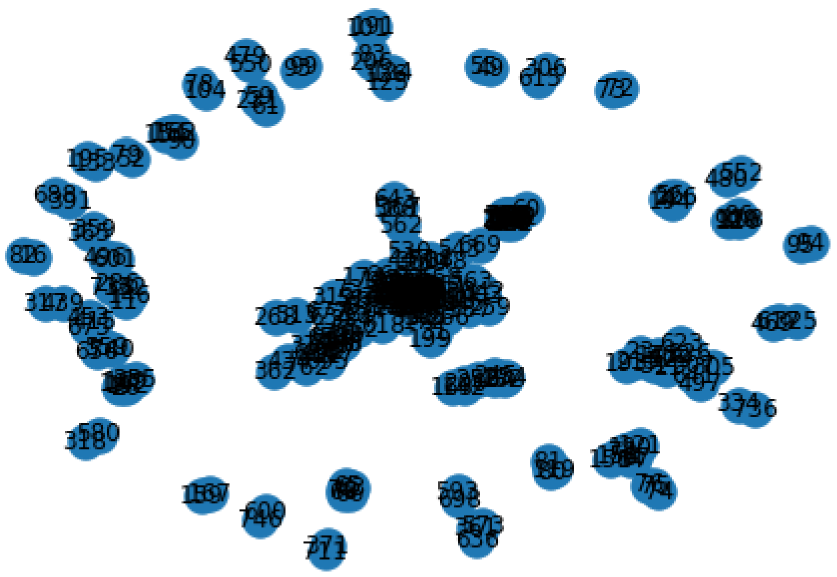


Graph drawing

After computing some indicators, let dra the graphs of relationship between user

Entrée [16]:

```
nt.draw(graph_user_user, with_labels=True)
```

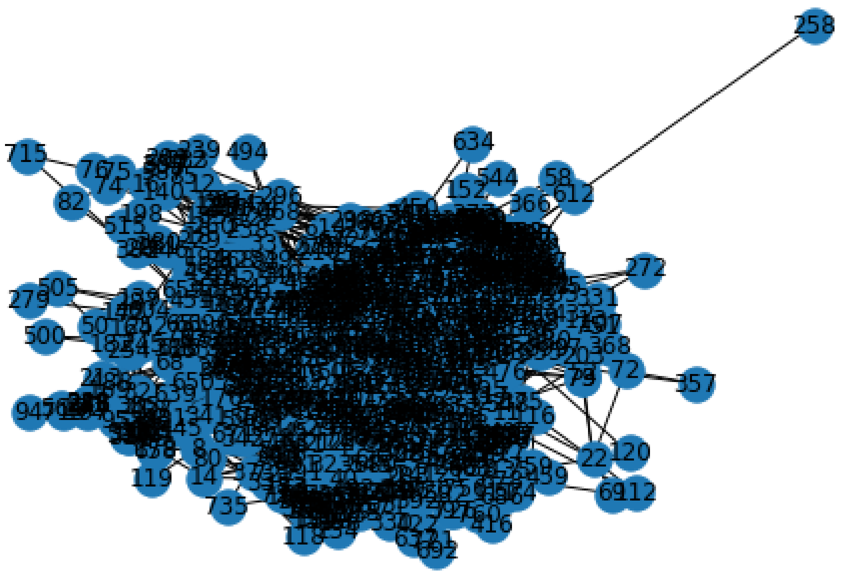


In the previous figure, we can notice that this graph are not connected.

Let draw graph for app and user interaction

Entrée [17]:

```
nt.draw(graph_user_app, with_labels=True)
```



Let compute the degree of centrality of each node

Entrée [18]:

```
degree_dict_user_user = nt.degree_centrality(graph_user_user)
degree_dict_user_app = nt.degree_centrality(graph_user_app)
```

After computing the degree of centrality of each

Entrée [19]:

```
print("The mean of the degree of centrality for the user relation graph is {}".form
print("The mean of the degree of centrality for the user interaction with app graph
```

```
The mean of the degree of centrality for the user relation graph is 0.
030785195434105843
The mean of the degree of centrality for the user interaction with app
graph is 0.006879742375604658
```

Now, we will compute this mean with this mean for only the spammer and only for the non-spammer

Entrée [20]:

```

list_to_compute_mean =[]
for node in degree_dict_user_user:
    if labels_dict['label'].get(node) is not None:
        if labels_dict['label'].get(node) ==1:
            list_to_compute_mean.append(degree_dict_user_user[node])
print(" The mean of the centrality of spammer : {}".format(np.mean(list_to_compute_



list_to_compute_mean =[]
for node in degree_dict_user_user:
    if labels_dict['label'].get(node) is not None:
        if labels_dict['label'].get(node) ==0:
            list_to_compute_mean.append(degree_dict_user_user[node])
print(" The mean of the centrality of non spammer : {}".format(np.mean(list_to_comp

```

```
 The mean of the centrality of spammer : 0.032967032967032967
 The mean of the centrality of non spammer : 0.0312184279652455
```

The mean of centrality of the spammer is pretty greater than the non spammer.

Computing of the betweenness of each node

More easily, this metric will show the best_connector

Entrée [21]:

```
betweeness_user_user = nt.betweenness_centrality(graph_user_user)
```

Entrée [22]:

```
betweeness_user_user = dict(sorted(betweeness_user_user.items(), key=lambda item: i
```

Let compute the mean of betweeness by class of node.

Entrée [23]:

```


list_to_compute_mean =[]
for node in betweeness_user_user:
    if labels_dict['label'].get(node) is not None:
        if labels_dict['label'].get(node) ==1:
            list_to_compute_mean.append(betweeness_user_user[node])
print(" The mean of the betweeness of spammer : {}".format(np.mean(list_to_compute_



list_to_compute_mean =[]
for node in betweeness_user_user:
    if labels_dict['label'].get(node) is not None:
        if labels_dict['label'].get(node) ==0:
            list_to_compute_mean.append(betweeness_user_user[node])
print(" The mean of the betweeness of non spammer : {}".format(np.mean(list_to_comp
```

```
 The mean of the betweeness of spammer : 0.0016196661726249728
 The mean of the betweeness of non spammer : 0.001884250561132695
```

Computing of average of clustering coefficient

Entrée [24]:

```
average_clustering_user_user = nt.average_clustering(graph_user_user)
average_clustering_user_app = nt.average_clustering(graph_user_app)
print ("The average of clustering coefficient of graph user relation is {}".format(
print ("The average of clustering coefficient of graph user app is {}".format(avera
```

```
The average of clustering coefficient of graph user relation is 0.4783
7313796999104
The average of clustering coefficient of graph user app is 0.176561478
36380048
```

From the previous indicators, the graph on user relationship is mre clustable than the other graph.

Entrée [25]:

```
closeness_user_user = nt.closeness_centrality(graph_user_user)
```

Entrée [26]:

```
closeness_user_user = dict(sorted(closeness_user_user.items(), key=lambda item: ite
```

Entrée [27]:

```
list_to_compute_mean_spam =[]
for node in closeness_user_user:
    if labels_dict['label'].get(node) is not None:
        if labels_dict['label'].get(node) ==1:
            list_to_compute_mean_spam.append(closeness_user_user[node])
list_to_compute_mean_non_spam =[]
for node in closeness_user_user:
    if labels_dict['label'].get(node) is not None:
        if labels_dict['label'].get(node) ==0:
            list_to_compute_mean_non_spam.append(closeness_user_user[node])
print("the mean of clossness of spammmer {}",np.mean(list_to_compute_mean_spam))
print("the mean of clossness of non spammmer {}",np.mean(list_to_compute_mean_non_s
```

```
the mean of clossness of spammmer {} 0.08221274876513192
the mean of clossness of non spammmer {} 0.08526386823516259
```

Link analysis

Analysis of the influence. The interpretation of this metric is influence of on node in the nerwork. At the end, we will display orded in most influenced node

Entrée [28]:

```
most_influential_link = nt.degree_centrality(graph_user_user)
```

Entrée [29]:

```
#Adding the label
for item in most_influential_link:
    most_influential_link[item]=(most_influential_link[item],labels_dict['label'].g
```

Page rank

Entrée [30]:

```
page_rank=nt.pagerank(graph_user_user)
```

Entrée [31]:

```
# the max page_rank
import operator
max_page_rank =max(page_rank.items(), key=operator.itemgetter(1))[0]
```

Let show the most page_rank node label

Entrée [32]:

```
labels_dict['label'].get(max_page_rank)
```

Out[32]:

```
1
```

Let compute the most important connection and its label. A node is high eigenvector centrality if it is connected to many other nodes who are themselves well connected.

Entrée [33]:

```
most_important_linked = nt.eigenvector_centrality(graph_user_user)
count =0
for w in sorted(most_important_linked, key=most_important_linked.get, reverse=True)
    print(w, most_important_linked[w],labels_dict['label'].get(w))
    count =count +1
    if count ==10:
        break
```

```
240 0.2053466952222802 0
137 0.20007116848212567 1
33 0.19154650519803795 0
136 0.1909179357115181 1
257 0.18913960441815067 0
223 0.1854685477854571 0
267 0.184522637831872 0
463 0.18053552510982565 0
241 0.18025828236593705 0
17 0.17975094125267083 0
```

Detection of community with girvan_newman on user relationship grah

Entrée [34]:

```
from networkx.algorithms import community
communities_generator = community.girvan_newman(graph_user_user)
top_level_communities = next(communities_generator)
next_level_communities = next(communities_generator)
coun = 0
print("The number of community with girvan_newman is : ", len(sorted(map(sorted, ne
for arr in sorted(map(sorted, next_level_communities)):
    #print("In the community {} there are {} there are follwoing node :".format(cou
    for node in arr:
        pass
        #print(node)
    coun =coun+1
```

```
The number of community with girvan_newman is :  47
```

Generation of graph bipartite

Entrée [35]:

```
from networkx.algorithms.community.kernighan_lin import kernighan_lin_bisection

print(kernighan_lin_bisection(graph_user_user))
```

```
({513, 8, 10, 11, 17, 19, 533, 24, 25, 26, 30, 31, 33, 34, 547, 36, 3
8, 41, 45, 49, 562, 53, 568, 57, 56, 572, 61, 573, 584, 73, 72, 75, 7
6, 77, 78, 593, 82, 84, 86, 87, 600, 89, 90, 93, 94, 607, 98, 99, 102,
615, 104, 106, 108, 109, 110, 623, 122, 634, 129, 643, 133, 648, 136,
138, 656, 146, 147, 660, 149, 662, 156, 669, 158, 672, 162, 163, 165,
168, 170, 687, 179, 180, 182, 698, 699, 188, 187, 190, 191, 192, 193,
194, 195, 197, 200, 201, 205, 220, 223, 736, 737, 227, 740, 231, 232,
233, 746, 236, 238, 239, 240, 241, 245, 247, 761, 762, 254, 257, 260,
262, 264, 265, 267, 286, 293, 305, 306, 307, 310, 317, 318, 319, 321,
325, 334, 338, 359, 361, 371, 377, 403, 407, 415, 424, 435, 480, 497},
{1, 3, 4, 516, 9, 16, 529, 18, 530, 532, 20, 21, 23, 27, 543, 35, 550,
552, 42, 43, 47, 48, 559, 51, 563, 52, 55, 570, 59, 60, 63, 64, 65, 6
6, 67, 68, 580, 70, 71, 74, 588, 79, 80, 81, 83, 601, 91, 92, 95, 101,
103, 105, 617, 618, 114, 119, 121, 636, 125, 126, 124, 641, 130, 132,
646, 137, 142, 654, 144, 145, 151, 154, 155, 668, 159, 160, 675, 164,
677, 166, 167, 169, 171, 172, 173, 174, 688, 183, 696, 185, 695, 196,
198, 199, 711, 204, 718, 206, 208, 207, 211, 214, 215, 218, 219, 738,
226, 228, 229, 237, 244, 250, 251, 252, 255, 259, 261, 266, 268, 269,
280, 282, 295, 309, 315, 326, 339, 342, 350, 362, 365, 367, 384, 387,
391, 411, 434, 439, 444, 447, 448, 453, 462, 463, 465, 469, 479, 496})
```

Detection of community with k-clique

With k=3, we will detect the communities

Entrée [36]:

```
communities_generator = nt.community.k_clique_communities(graph_user_user, 3)
top_level_communities = next(communities_generator)
next_level_communities = next(communities_generator)
print(next_level_communities)
```

```
frozenset({56, 194, 226, 4})
```

Clustering based on simularity

Now let'use K-means to do clustering. This part is inspired from [link (https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_digits.html)](https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_digits.html)

We will use k-means with differents configuration after that, we will compute some metric to evalute the cluster

Entrée [37]:

```
filepath ="./data/Social_spammers_dataset/graphs/app_based_similarity/sim_matrix.tx
matrix = open(filepath).read()
matrix = [item.split(",") for item in matrix.split('\n')[:-1]]
```

Entrée [38]:

```
matrix_np = np.array(matrix)
matrix_np = matrix_np.astype(np.float32)
```

Entrée [39]:

```
columns = [ index for index in range(1,768) ]
rows = [ index for index in range(1,768) ]
dataframe_from_sim_matrix = pd.DataFrame(data=matrix_np, index=rows, columns=column
```

Entrée [40]:

```
from sklearn.cluster import KMeans
```

Setting our benchmarck

This code allows us to make comparaision beetween the variants of k-means, and adding PCA

Entrée [41]:

```

def bench_k_means(kmeans, name, data, labels):
    t0 = time()
    estimator = make_pipeline(StandardScaler(), kmeans).fit(data)
    fit_time = time() - t0
    results = [name, fit_time, estimator[-1].inertia_]

    # Define the metrics which require only the true labels and estimator
    # labels
    clustering_metrics = [
        metrics.homogeneity_score,
        metrics.completeness_score,
        metrics.v_measure_score,
        metrics.adjusted_rand_score,
        metrics.adjusted_mutual_info_score,
    ]
    results += [m(labels, estimator[-1].labels_) for m in clustering_metrics]

    # The silhouette score requires the full dataset
    results += [
        metrics.silhouette_score(data, estimator[-1].labels_,
                                 metric="euclidean", sample_size=300,)
    ]

    # Show the results
    formatter_result = ("{:9s}\t{:.3f}s\t{:.0f}\t{:.3f}\t{:.3f}"
                        "\t{:.3f}\t{:.3f}\t{:.3f}\t{:.3f}")
    print(formatter_result.format(*results))
```

Entrée [42]:

```
def benchmark_k_mean(dataframe,labels):
    for i in range(2,10):
        print ("With K = {}".format(i))
        print(82 * '_')
        print('init\t\ttime\tinertia\thomo\tcompl\tv-meas\tARI\tAMI\tsilhouette')

        kmeans = KMeans(init="k-means++", n_clusters=i, n_init=4,
                        random_state=0)
        bench_k_means(kmeans=kmeans, name="k-means++", data=dataframe, labels=label

        kmeans = KMeans(init="random", n_clusters=i, n_init=4, random_state=0)
        bench_k_means(kmeans=kmeans, name="random", data=dataframe, labels=labels)

        pca = PCA(n_components=i).fit(dataframe)
        kmeans = KMeans(init=pca.components_, n_clusters=i, n_init=1)
        bench_k_means(kmeans=kmeans, name="PCA-based", data=dataframe, labels=label

        print(82 * '_')
```

The matrix from sim_matrix.txt

Entrée [43]:

```
dataframe_from_sim_matrix.shape
```

Out[43]:

```
(767, 767)
```

Clustering of the data from sim matrix

In this section we will make cluster(community) with the file.

Entrée [44]:

```
benchmark_k_mean(dataframe_from_sim_matrix,[index for index in range(1,768)])
```

```
With K = 2

__________________________________________________________________________________

______________
init            time    inertia homo    compl   v-meas  ARI     AMI
silhouette
k-means++       0.093s  378817  0.102   1.000   0.185   0.000   0.000
0.559
random          0.079s  378817  0.102   1.000   0.185   0.000   0.000
0.587
PCA-based       0.035s  378817  0.102   1.000   0.185   0.000   0.000
0.539

__________________________________________________________________________________

______________
With K = 3

__________________________________________________________________________________

______________
init            time    inertia homo    compl   v-meas  ARI     AMI
silhouette
k-means++       0.068s  291944  0.157   1.000   0.271   0.000   -0.000
0.528
random          0.051s  291944  0.157   1.000   0.271   0.000   -0.000
0.563
PCA-based       0.026s  291944  0.157   1.000   0.271   0.000   -0.000
0.551

__________________________________________________________________________________

______________
With K = 4

__________________________________________________________________________________

______________
init            time    inertia homo    compl   v-meas  ARI     AMI
silhouette
k-means++       0.070s  247854  0.189   1.000   0.318   0.000   0.000
0.533
random          0.044s  251112  0.188   1.000   0.317   0.000   0.000
0.543
PCA-based       0.025s  251112  0.188   1.000   0.317   0.000   0.000
0.551

__________________________________________________________________________________

______________
With K = 5

__________________________________________________________________________________

______________
init            time    inertia homo    compl   v-meas  ARI     AMI
silhouette
k-means++       0.080s  226393  0.208   1.000   0.345   0.000   0.000
0.505
random          0.055s  208387  0.219   1.000   0.359   0.000   0.000
0.573
PCA-based       0.028s  208386  0.218   1.000   0.359   0.000   -0.000
0.578

__________________________________________________________________________________

______________
With K = 6

__________________________________________________________________________________

______________
init            time    inertia homo    compl   v-meas  ARI     AMI
silhouette
```

```
k-means++       0.086s  192941  0.235   1.000   0.380   0.000   -0.000
0.566
random          0.055s  188108  0.237   1.000   0.383   0.000   0.000
0.582
PCA-based       0.026s  192941  0.235   1.000   0.380   0.000   -0.000
0.574
______________________________________________________________________
____________
With K = 7
______________________________________________________________________
____________
init            time    inertia homo    compl   v-meas  ARI     AMI
silhouette
k-means++       0.099s  173039  0.252   1.000   0.403   0.000   -0.000
0.587
random          0.069s  178700  0.259   1.000   0.412   0.000   -0.000
0.546
PCA-based       0.027s  178684  0.249   1.000   0.398   0.000   -0.000
0.511
______________________________________________________________________
____________
With K = 8
______________________________________________________________________
____________
init            time    inertia homo    compl   v-meas  ARI     AMI
silhouette
k-means++       0.116s  161308  0.263   1.000   0.417   0.000   -0.000
0.581
random          0.061s  162884  0.283   1.000   0.441   0.000   -0.000
0.577
PCA-based       0.026s  166754  0.260   1.000   0.412   0.000   -0.000
0.567
______________________________________________________________________
____________
With K = 9
______________________________________________________________________
____________
init            time    inertia homo    compl   v-meas  ARI     AMI
silhouette
k-means++       0.131s  155523  0.288   1.000   0.447   0.000   0.000
0.518
random          0.064s  153667  0.284   1.000   0.442   0.000   -0.000
0.568
PCA-based       0.027s  147732  0.276   1.000   0.433   0.000   -0.000
0.630
______________________________________________________________________
____________
```

From the matrix file, we made clusters (communities). For the K=9, that is the best number of K according to the silhouette metric

Now we will do the clustering with the matrix from the graph user interaction

We also apply a clustering algorithm (k-means, hierarchical algorithms). we also generated the matrix with simarilty from graph of user relationship.

Entrée [45]:

```
graph_user_user_matrix = nt.adjacency_matrix(graph_user_user,weight='Sim')
```

Entrée [46]:

```
print(graph_user_user_matrix.todense())
labels = graph_user_user.nodes()
graph_user_user_matrix = pd.DataFrame(graph_user_user_matrix.todense(), index=list(
```

```
[[0.    0.975 0.    ... 0.    0.    0.   ]
 [0.975 0.    0.    ... 0.    0.    0.   ]
 [0.    0.    0.    ... 0.    0.    0.   ]
 ...
 [0.    0.    0.    ... 0.    0.    0.   ]
 [0.    0.    0.    ... 0.    0.    0.   ]
 [0.    0.    0.    ... 0.    0.    0.   ]]
```

Let show the shape on second matrix

Entrée [47]:

```
graph_user_user_matrix.shape
```

Out[47]:

```
(295, 295)
```

Entrée [48]:

```
graph_user_user_matrix
```

Out[48]:

	1	**168**	**4**	**56**	**149**	**244**	**198**	**48**	**25**	**232**	**...**	**167**	**365**	**359**
1	0.000	0.975	0.000	0.000	1.000	1.000	1.000	1.000	0.999	0.943	...	0.0	0.0	0.0
168	0.975	0.000	0.000	0.000	0.971	0.971	0.971	0.971	0.982	0.982	...	0.0	0.0	0.0
4	0.000	0.000	0.000	0.974	0.000	0.000	0.000	0.000	0.000	0.000	...	0.0	0.0	0.0
56	0.000	0.000	0.974	0.000	0.000	0.000	0.000	0.000	0.000	0.000	...	0.0	0.0	0.0
149	1.000	0.971	0.000	0.000	0.000	1.000	1.000	1.000	0.998	0.936	...	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
439	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	...	0.0	0.0	0.0
317	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	...	0.0	0.0	0.0
174	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	...	0.0	0.0	0.0
677	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	...	0.0	0.0	0.0
193	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	...	0.0	0.0	0.0

295 rows × 295 columns

Entrée [49]:

```
benchmark_k_mean(graph_user_user_matrix,graph_user_user.nodes())
```

```
With K = 2
_______________________________________________________________________
____________
init            time    inertia homo    compl   v-meas  ARI     AMI
silhouette
k-means++       0.034s  79735   0.059   1.000   0.112   0.000   -0.000
0.493
random          0.022s  79735   0.059   1.000   0.112   0.000   -0.000
0.493
PCA-based       0.021s  79735   0.059   1.000   0.112   0.000   -0.000
0.493
_______________________________________________________________________
____________
With K = 3
_______________________________________________________________________
____________
init            time    inertia homo    compl   v-meas  ARI     AMI
silhouette
k-means++       0.033s  73186   0.106   1.000   0.191   0.000   -0.000
0.481
random          0.024s  78451   0.076   1.000   0.142   0.000   -0.000
0.095
PCA-based       0.012s  73171   0.107   1.000   0.193   0.000   -0.000
0.481
_______________________________________________________________________
____________
With K = 4
_______________________________________________________________________
____________
init            time    inertia homo    compl   v-meas  ARI     AMI
silhouette
k-means++       0.026s  71016   0.154   1.000   0.267   0.000   -0.000
0.447
random          0.021s  76170   0.129   1.000   0.229   0.000   -0.000
0.138
PCA-based       0.011s  71009   0.161   1.000   0.277   0.000   -0.000
0.424
_______________________________________________________________________
____________
With K = 5
_______________________________________________________________________
____________
init            time    inertia homo    compl   v-meas  ARI     AMI
silhouette
k-means++       0.028s  70059   0.174   1.000   0.297   0.000   -0.000
0.127
random          0.019s  76395   0.118   1.000   0.210   0.000   -0.000
0.051
PCA-based       0.010s  70086   0.191   1.000   0.320   0.000   -0.000
0.367
_______________________________________________________________________
____________
With K = 6
_______________________________________________________________________
____________
init            time    inertia homo    compl   v-meas  ARI     AMI
silhouette
```

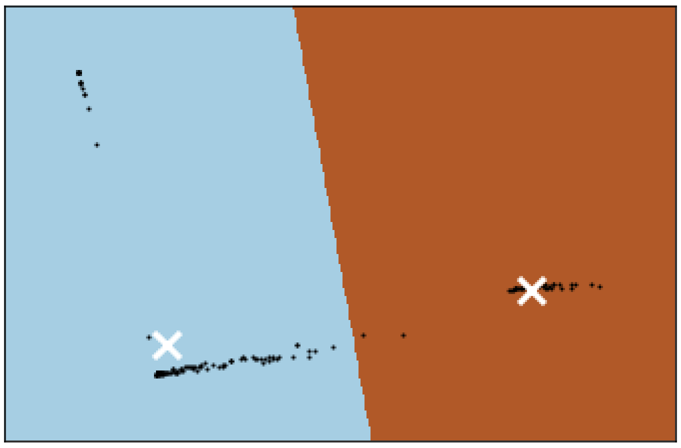
```
k-means++       0.027s  68877   0.188   1.000   0.317   0.000   -0.000
0.136
random          0.025s  75790   0.127   1.000   0.226   0.000   -0.000
-0.103
PCA-based       0.014s  68864   0.203   1.000   0.337   0.000   -0.000
0.217
______________________________________________________________________
____________
With K = 7
______________________________________________________________________
____________
init            time    inertia homo    compl   v-meas  ARI     AMI
silhouette
k-means++       0.029s  68324   0.198   1.000   0.330   0.000   -0.000
0.104
random          0.023s  73679   0.177   1.000   0.301   0.000   -0.000
-0.063
PCA-based       0.012s  68440   0.219   1.000   0.359   0.000   -0.000
0.218
______________________________________________________________________
____________
With K = 8
______________________________________________________________________
____________
init            time    inertia homo    compl   v-meas  ARI     AMI
silhouette
k-means++       0.034s  68672   0.151   1.000   0.262   0.000   -0.000
-0.020
random          0.021s  73223   0.163   1.000   0.280   0.000   -0.000
-0.078
PCA-based       0.011s  67831   0.227   1.000   0.370   0.000   -0.000
0.217
______________________________________________________________________
____________
With K = 9
______________________________________________________________________
____________
init            time    inertia homo    compl   v-meas  ARI     AMI
silhouette
k-means++       0.031s  67201   0.207   1.000   0.342   0.000   -0.000
0.127
random          0.022s  72923   0.167   1.000   0.286   0.000   -0.000
-0.157
PCA-based       0.012s  67339   0.254   1.000   0.405   0.000   -0.000
0.214
______________________________________________________________________
____________
```

The best silhouette is obtained when we uses k as 2 . Let draw to get more insight about these clusters.

Entrée [50]:

```
import matplotlib.pyplot as plt

reduced_data = PCA(n_components=2).fit_transform(graph_user_user_matrix)
kmeans = KMeans(init="k-means++", n_clusters=2, n_init=4)
kmeans.fit(reduced_data)

h = .02
x_min, x_max = reduced_data[:, 0].min() - 1, reduced_data[:, 0].max() + 1
y_min, y_max = reduced_data[:, 1].min() - 1, reduced_data[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

# Obtain labels for each point in mesh. Use last trained model.
Z = kmeans.predict(np.c_[xx.ravel(), yy.ravel()])

# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.figure(1)
#plt.figure(figsize=(200,100))
plt.clf()
plt.imshow(Z, interpolation="nearest",
           extent=(xx.min(), xx.max(), yy.min(), yy.max()),
           cmap=plt.cm.Paired, aspect="auto", origin="lower")

plt.plot(reduced_data[:, 0], reduced_data[:, 1], 'k.', markersize=2)
# Plot the centroids as a white X
centroids = kmeans.cluster_centers_
plt.scatter(centroids[:, 0], centroids[:, 1], marker="x", s=169, linewidths=3,
            color="w", zorder=10)
plt.title("K-means clustering user (PCA-reduced data)\n"
          "Centroids are marked with white cross")
plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
plt.xticks(())
plt.yticks(())

plt.show()
```

K-means clustering user (PCA-reduced data)
Centroids are marked with white cross



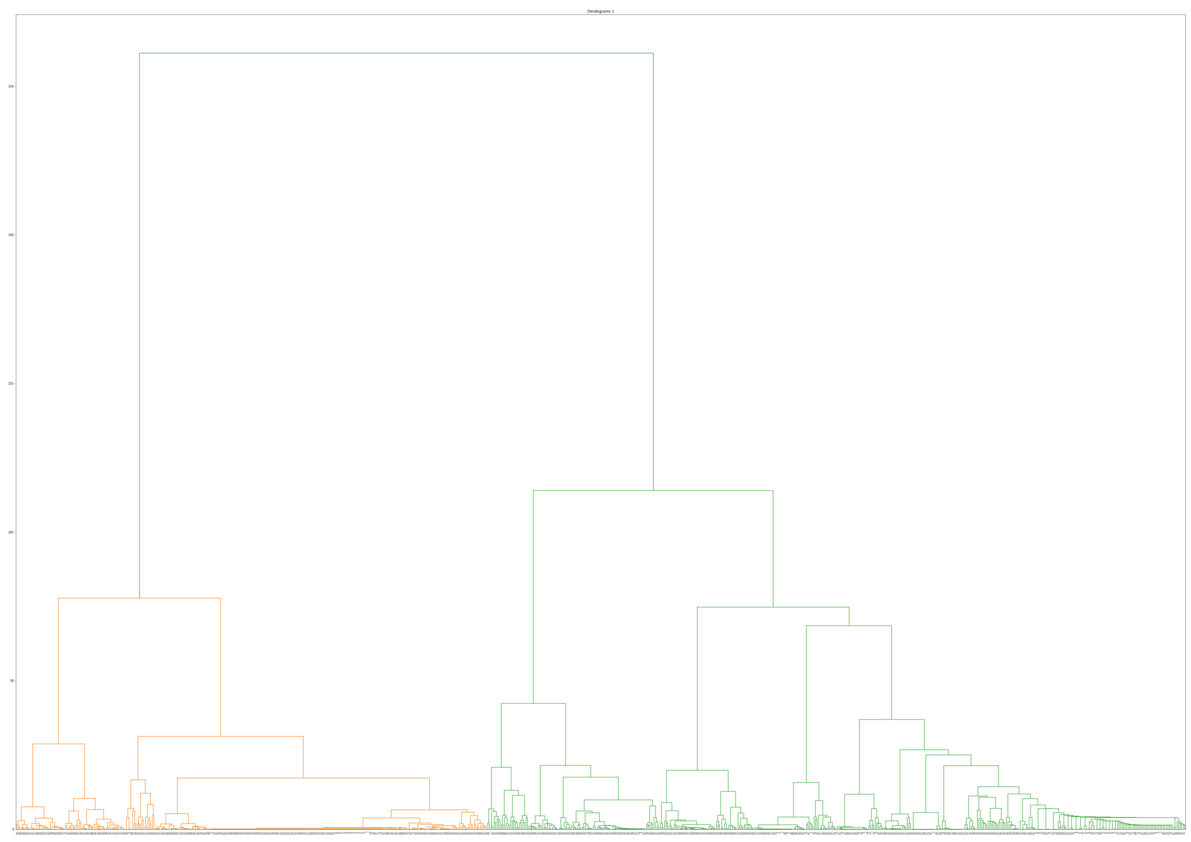
Hierarchical Clustering

To end this study let uild the dendogram from Hierarchical Clustering

The following section will compute the cluster from the graph matrix of simularity. From the previous section the right number of cluster is 2 for this matrix

Entrée [51]:

```
import scipy.cluster.hierarchy as shc

plt.figure(figsize=(70, 50))
plt.title("Dendograms 1")
dend = shc.dendrogram(shc.linkage(dataframe_from_sim_matrix, method='ward'))
```



This section will compute the cluter from the sim_matrix.txt file.

Let's build the dendogram before make cluster.

According to the dendogram, the value 2 is right for the number of cluster.

Entrée [52]:

```
from sklearn.cluster import AgglomerativeClustering

cluster_AgglomerativeClustering = AgglomerativeClustering(n_clusters=2, affinity='e
cluster_AgglomerativeClustering.fit_predict(dataframe_from_sim_matrix)
```

Out[52]:

```
array([0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0,
0,
       0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1,
0,
       1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
1,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0,
0,
       0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
0,
       0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0,
0,
       1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0,
0,
       0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1,
1,
       0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1,
1,
       0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0,
1,
       1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1,
0,
       0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0,
1,
       1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0,
1,
       0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0,
1,
       0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0,
1,
       0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0,
0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0,
0,
       1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1,
0,
       1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0,
0,
       0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0,
0,
       1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0,
0,
       0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0,
0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0,
1,
        1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1,
1,
        0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1,
0,
        1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0,
1,
        0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0,
1,
        0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0,
1,
        0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1,
0,
        0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1,
0,
        0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0,
1,
        0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0])
```

We call add this group to the dataset for the prediction task for the spammer. We think that can improve the accuracy of model.

Entrée [ ]:

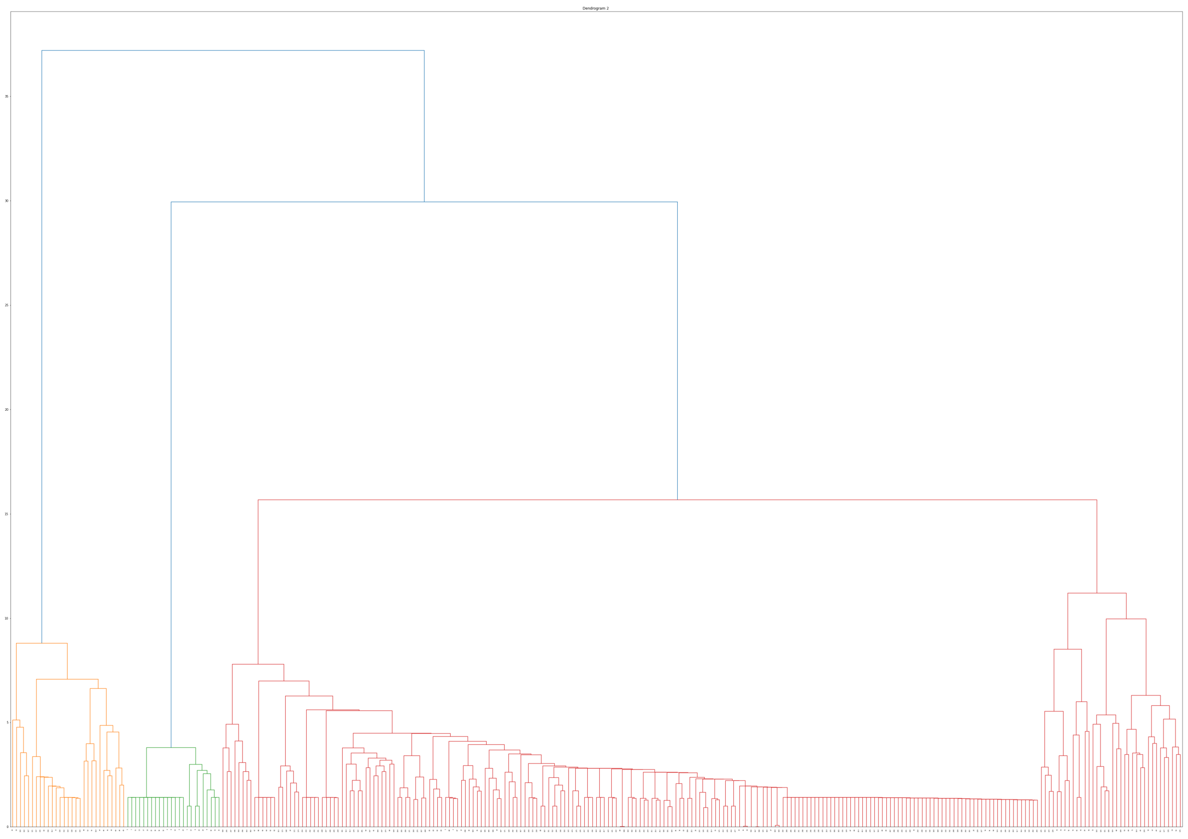
```

```

Let build the dendogram of this two dataset

Entrée [53]:

```
import scipy.cluster.hierarchy as shc

plt.figure(figsize=(70, 50))
plt.title("Dendrogram 2 ")
dend = shc.dendrogram(shc.linkage(graph_user_user_matrix, method='ward'))
```

```
<ipython-input-53-7f43d5beea76>:5: ClusterWarning: scipy.cluster: The
symmetric non-negative hollow observation matrix looks suspiciously li
ke an uncondensed distance matrix
  dend = shc.dendrogram(shc.linkage(graph_user_user_matrix, method='wa
rd'))
```



Entrée [54]:

```
from sklearn.cluster import AgglomerativeClustering

cluster_AgglomerativeClustering = AgglomerativeClustering(n_clusters=3, affinity='e
cluster_AgglomerativeClustering.fit_predict(graph_user_user_matrix)
```

```
/home/abou/anaconda3/envs/tf/lib/python3.8/site-packages/scipy/cluste
r/hierarchy.py:826: ClusterWarning: scipy.cluster: The symmetric non-n
egative hollow observation matrix looks suspiciously like an uncondens
ed distance matrix
  return linkage(y, method='ward', metric='euclidean')
```

Out[54]:

```
array([2, 2, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2,
       2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
1,
       1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0,
0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0,
0,
       0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0])
```