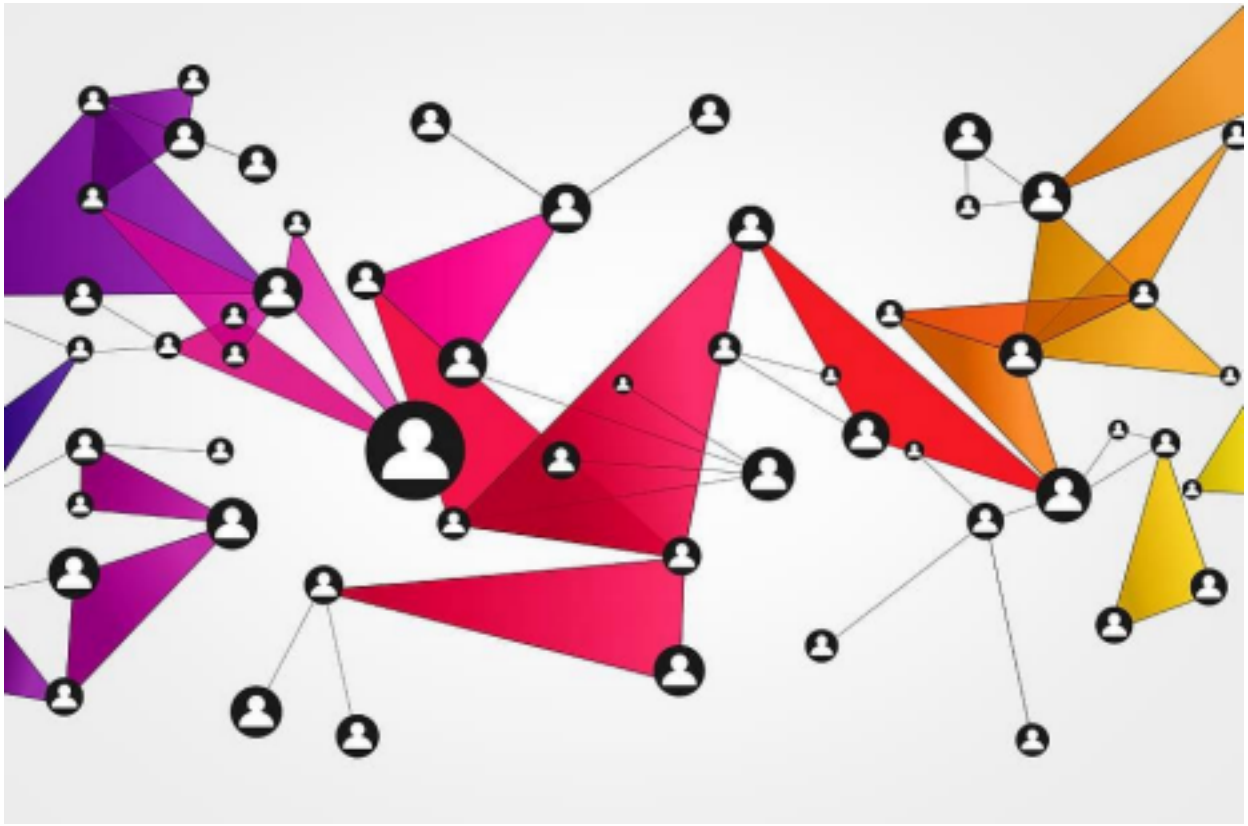
Group Project



19OH01 - Social and Economic Network Analysis

Topic: ENRON E-mail analysis

19Z328-Lavanya Ra

19Z320 - Harshitha P

19Z348 - Sravya V

19Z352 - Sushmitha S

19Z354 - Swetha G N

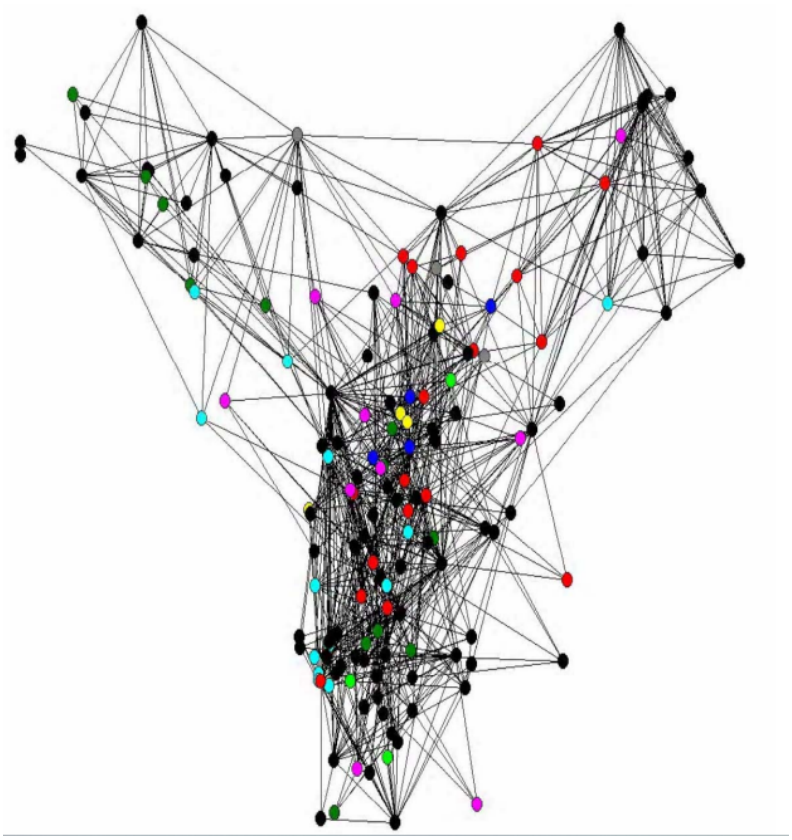
PROBLEM STATEMENT

One of the biggest corporate collapses in history was caused by the Enron scandal and collapse. One of the biggest energy firms in America in 2000 was Enron. After being exposed for fraud, it then plummeted into bankruptcy in less than a year. We have access to the 500 000 emails sent and received between 150 former Enron workers, most of whom were top executives, in the Enron email database. Its value is increased by the fact that it is the only sizable public database of authentic emails. In reality, data scientists have been studying and researching with this dataset for years.

DATASET DESCRIPTION

The CALO Project assembled and produced this dataset (A Cognitive Assistant that Learns and Organizes). It has information arranged into folders from roughly 150 users, most of whom are high managers at Enron. The corpus has roughly 0.5 million messages in it. The Federal Energy Regulatory Commission initially made this information available to the public and put it online while conducting its inquiry. The collection consists of 150 individuals' 517,431 mails, primarily top management from the Enron Corporation. Despite the size of the collection, there are frequently few thematic folders for certain users. We disregard everyone's inboxes and only review emails that have already been sent for our purposes.
By using this strategy, we can prevent unintentionally analyzing the received emails that contain spam. Sentiment analysis and topic modeling with Latent Dirichlet Allocation (LDA) were the two main analytical techniques used.

The basic definition of a social contact between two employees is a pre-defined threshold number of exchange of emails.
We only evaluated bidirectional links, which implies that we consider a contact if both parties have sent emails to each other. This ensures that information was exchanged between the two Enron employees and that they were engaged in some kind of communication. To manifest the flow of information in an organization, we also considered each employee's position in the organizational hierarchy.



Link to download dataset - https://www.cs.cmu.edu/~./enron/

TOOLS USED

- Google Colab
  - Google Colab is a free Jupyter Notebook environment that runs entirely in Google Cloud. And utilizes the virtual GPU.
- NetworkX
  - NetworkX is a python graph library which is used to construct, manipulate and analyze the structure, features and metrics of the input graph.
- Pandas
  - DataFrame and Series are two sets of potent tools provided by The Pandas that are mostly utilized for data analysis.
- Matplotlib
  - This library is usually paired along with Numpy to allow users to use python to plot various visualizations.

CHALLENGES FACED.

- Visualization of a huge dataset. Since most of the nodes in the dataset are too relative to each other, it is difficult to view them as separate nodes on the screen.
- Data gathering techniques and scalability issues.
- Missing data verification and correct debugging of errors.
- Some modules were outdated, so to find a module that has proper documentation research had to be done.

CONTRIBUTION OF TEAM MEMBERS

Harshitha P - 19Z320	Worked on implementing Page Rank Algorithm and worked on calculating basic metrics of the network
Lavanya Ra - 19Z328	Worked on Implementing the HITS Algorithm, Worked on finding out the Central and Peripheral nodes
Sravya V - 19Z348	Worked on implementing community detection using Girvan–Newman Algorithm, Displaying the important nodes based on centrality
Sushmitha S - 19Z352	Worked on implementing community detection using Girvan–Newman Algorithm, Connectivity and disconnectivity of Network
Swetha G N - 19Z354	Worked on implementing community detection using Girvan–Newman Algorithm, checking whether the network is Strongly connected or bipartite graph

ANNEXURE I: CODE

Import necessary libraries

```
import pandas as pd
import networkx as nx
import numpy as np
import matplotlib.pyplot as plt

default_colors = [
    '#1f77b4',  '#ff7f0e',  '#2ca02c', '#d62728',  '#9467bd',   '#8c564b',  '#e377c2',
    '#7f7f7f',  '#bcbd22',  '#17becf',  '#e6194b', '#3cb44b',   '#ffe119',  '#4363d8',
    '#f58231',  '#911eb4', '#46f0f0','#f032e6',    '#bcf60c',    '#fabebe',    '#008080',    '#e6beff',
    '#9a6324','#fffac8',    '#800000',    '#aaffc3',    '#808000',    '#ffd8b1',    '#000075',
    '#808080',
]
```

Import dataset

```
from google.colab import files
uploaded = files.upload()
```

```
df = pd.read_table('./email-enron-only.mtx', header= None, sep= ' ')
df.columns = ['source','target']
df.head(5)
```

Basic metrics and finding if its a connected graph and if it is a bipartite graph

```
print("Max node degree (in + out) is {}".format(max(nx.degree(graph), key= lambda x: x[1])[1]))
print("Max node degree (in + out) is {}".format(min(nx.degree(graph), key= lambda x: x[1])[1]))
print("Diameter of the network is {}".format(nx.diameter(graph)))
print("Radius of the network is {}".format(nx.radius(graph)))
print("Average distance in the network {}\n".format(nx.average_shortest_path_length(graph)))

print("Is bipartite? {}".format(nx.is_bipartite(graph)))
print("Is connected? {}".format(nx.is_connected(graph)))
```

Central and periphery nodes

```
def colors(n,l):
    if n in l:
        return 'red'
    else:
        return '#1f78b4'

centers = nx.center(graph)
node_color = [colors(n,centers) for n in graph.nodes()]

print("Centers of the network: ",centers)

plt.figure(figsize=(15,10))
nx.draw_networkx(
    graph,
    with_labels= False,
    alpha= 0.6,
    edge_color= '0.85',
    edge_size= 5,
    node_size = 400,
    node_color= node_color,
    pos= pos
)
plt.axis('off')
plt.title("Center node in the Network", fontdict= {'size':15})
```

```
def colors(n,l):
    if n in l:
        return 'red'
    else:
        return '#1f78b4'

centers = nx.center(graph)
node_color = [colors(n,centers) for n in graph.nodes()]

print("Centers of the network: ",centers)

plt.figure(figsize=(15,10))
nx.draw_networkx(
    graph,
    with_labels= False,
    alpha= 0.6,
    edge_color= '0.85',
    edge_size= 5,
    node_size = 400,
    node_color= node_color,
    pos= pos
)
plt.axis('off')
plt.title("Center node in the Network", fontdict= {'size':15})
```

```
periphery = nx.periphery(graph)
node_color = [colors(n,periphery) for n in graph.nodes()]

print("Peripheries of the network: ",periphery)

plt.figure(figsize=(15,10))
nx.draw_networkx(
    graph,
    with_labels= False,
    alpha= 0.6,
    edge_color= '0.85',
    edge_size= 5,
    node_size= 400,
    node_color= node_color,
    pos= pos
)
plt.axis('off')
plt.title("Periphery nodes in Network", fontdict= {'size':15})
```

Disconnectivity of graph

```
print("Number of node removals - {}, the node - {}".format(nx.node_connectivity(graph), nx.minimum_node_cut(graph)))
print("Number of edge removals - {}, the edge - {}".format(nx.edge_connectivity(graph), nx.minimum_edge_cut(graph)))
```

```
node_conn = nx.minimum_node_cut(graph)
edge_conn = nx.minimum_edge_cut(graph)
node_color = [colors(n,node_conn) for n in graph.nodes()]
edge_color = ['0.85' if e not in edge_conn else 'r' for e in graph.edges()]

plt.figure(figsize=(15,10))
nx.draw_networkx(
    graph,
    with_labels= False,
    alpha= 0.6,
    edge_color= edge_color,
    edge_size= 5,
    node_size= 400,
    node_color= node_color,
    pos= pos
)
nx.draw_networkx_edges(graph, edgelist= edge_conn, pos= pos, edge_color= 'r', width= 3, alpha= 0.6)
plt.axis('off')
plt.title("Node and Edge connectivity", fontdict= {'size':15})
```

Clustering coefficient

```
print("Average clustering coefficient {}".format(nx.average_clustering(graph)))
print("Transitivity of the network {}".format(nx.transitivity(graph)))
```

Degree Centrality

```
deg_centrality = nx.degree_centrality(graph)
size = [deg_centrality[n]*2500 for n in graph.nodes()]
color = [graph.degree(n) for n in graph.nodes()]

plt.figure(figsize= (15,10))
nx.draw_networkx(
    graph,
    with_labels= False,
    edge_color= '0.4',
    width= 0.1,
    node_size= size,
    node_color= color,
    alpha= 0.9,
    pos= pos,
    cmap= plt.cm.Reds
)
plt.axis('off')
plt.title("Node importance as per Degree centality", fontdict= {'size':15})

print("10 most important nodes as per Degree centrality\n {}\n\n".format(
    [n[0] for n in sorted(deg_centrality.items(), key= lambda x: x[1], reverse= True)[:10]]
))
```

Betweenness Centrality

```
betweenness_centrality = nx.betweenness_centrality(graph)
size = [betweenness_centrality[n]*5000 for n in graph.nodes()]
color = [betweenness_centrality[n] for n in graph.nodes()]

plt.figure(figsize= (15,10))
nx.draw_networkx(
    graph,
    with_labels= False,
    edge_color= '0.4',
    width= 0.1,
    node_size= size,
    node_color= color,
    alpha= 0.8,
    pos= pos,
    cmap= plt.cm.Reds
)
plt.axis('off')
plt.title("Node importance as per Betweenness centrality", fontdict= {'size':15})

print("10 most important nodes as per Betweenness centrality\n {}\n\n".format(
    [n[0] for n in sorted(betweenness_centrality.items(), key= lambda x: x[1], reverse= True)[:10]]
))
```

Closeness Centrality

```
closeness_centrality = nx.closeness_centrality(graph)
size = [-1000 * np.log((1 - closeness_centrality[n]))**3 for n in graph.nodes()]
color = [closeness_centrality[n] for n in graph.nodes()]

plt.figure(figsize= (15,10))
nx.draw_networkx(
    graph,
    with_labels= False,
    edge_color= '0.4',
    width= 0.1,
    node_size= size,
    node_color= color,
    alpha= 0.9,
    pos= pos,
    cmap= plt.cm.Reds
)
plt.axis('off')
plt.title("Node importance as per Closeness centrality", fontdict= {'size':15})

print("10 most important nodes as per Closeness centrality\n {}\n\n".format(
    [n[0] for n in sorted(closeness_centrality.items(), key= lambda x: x[1], reverse= True)[:10]]
))
```

Page Rank

```
digraph = nx.from_pandas_edgelist(df, create_using= nx.DiGraph)
page_ranks = nx.pagerank(digraph)
size = [page_ranks[n]*11000 for n in digraph.nodes()]
color = [page_ranks[n] for n in digraph.nodes()]

plt.figure(figsize=(15,10))
nx.draw_networkx(
    digraph,
    alpha= 0.9,
    with_labels= False,
    node_size= size,
    node_color= color,
    edge_color= '0.85',
    width= 0.5,
    pos= pos,
    cmap= plt.cm.Reds
)
plt.axis('off')
plt.title("Node importance as per Page Rank", fontdict= {'size':15})

print("10 most important nodes as per Page Rank\n {}\n\n".format(
    [n[0] for n in sorted(page_ranks.items(), key= lambda x: x[1], reverse= True)[:10]]
))
```

HITS Algorithm

```
hubs_and_authority = nx.hits(digraph)
hubs = hubs_and_authority[0]
auth = hubs_and_authority[1]
size = [hubs[n] * 11000 if hubs[n] >= auth[n] else auth[n] * 11000 for n in digraph.nodes()]
color = ['#1f78b4' if hubs[n] > auth[n] else 'r' if hubs[n] < auth[n] else 'grey' for n in digraph.nodes()]

plt.figure(figsize=(15,10))
nx.draw_networkx(
    digraph,
    alpha= 0.6,
    with_labels= False,
    node_size= size,
    node_color= color,
    edge_color= '0.85',
    width= 0.5,
    pos= pos,
    cmap= plt.cm.Reds
)
plt.axis('off')
plt.title("Node importance as per Hub and Authority scores", fontdict= {'size':15})

print("10 most important nodes as per Hub scores\n {}\n".format(
    [n[0] for n in sorted(hubs.items(), key= lambda x: x[1], reverse= True)[:10]]
))
print("10 most important nodes as per Authority scores\n {}\n\n".format(
    [n[0] for n in sorted(auth.items(), key= lambda x: x[1], reverse= True)[:10]]
))
```

Girvan-Newman algorithm

```
def edge_to_remove(graph):
    edge_btw_centrality = nx.edge_betweenness_centrality(graph)
    edge = max(edge_btw_centrality.items(), key= lambda item: item[1])[0]

    return edge

def girvan_newman(graph,k):
    conn_comps = nx.connected_components(graph)
    num_conn_comps = nx.number_connected_components(graph)

    while(num_conn_comps < k):
        edge = edge_to_remove(graph)
        graph.remove_edge(edge[0],edge[1])
        conn_comps = nx.connected_components(graph)
        num_conn_comps = nx.number_connected_components(graph)

    return conn_comps

def color_grps(n):
    if n in node_grps[0]:
        return 'b'
    elif n in node_grps[1]:
        return 'r'
    else:
        return 'g'

g = girvan_newman(graph.copy(),3)
node_grps = []

for i in list(g):
    node_grps.append(i)

color = [color_grps(n) for n in graph.nodes()]

plt.figure(figsize=(15,10))
nx.draw_networkx(
    graph,
    alpha= 0.6,
    node_size= 400,
    node_color= color,
    edge_color= '0.85',
    width = 0.5,
    with_labels= False
)
plt.axis('off')
plt.title('Communities within the network', fontdict= {'size':15})
```

Identifying optimal Community split

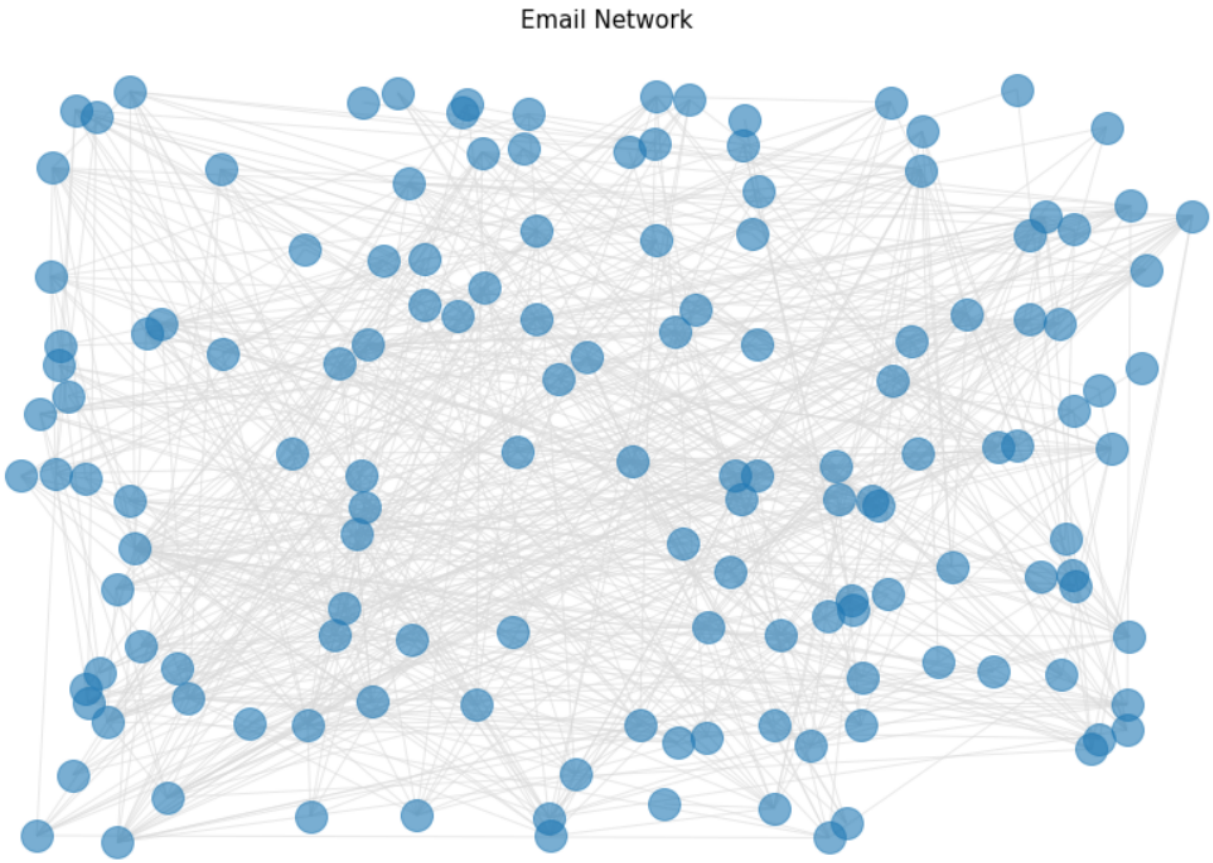
```
g = nx.community.girvan_newman(graph.copy())
comm_grp = []

for i in list(g):
    comm_grp.append(i)

mod_scores = [nx.community.modularity(graph,grp) for grp in comm_grp]
perf = [nx.community.performance(graph, grp) for grp in comm_grp]
clusters = [x for x in range(2,len(comm_grp[-1])+1)]

plt.figure(figsize=(12,8))
plt.plot(clusters, mod_scores, label= 'Modularity', color= 'r')
plt.plot(clusters, perf, label= 'Performance', color = 'b')
plt.axvline(x= clusters[np.argmax(mod_scores)], color= 'grey', linestyle= '--', alpha= 0.6)
plt.annotate(
    xy=(clusters[np.argmax(mod_scores)]+1,0.05),
    s='Cluster size: {}\nModularity: {}\nPerformance: {}'.format(
        clusters[np.argmax(scores)],
        round(max(mod_scores),2),
        round(max(perf),2)
))
plt.title("Modularity and Performance scores vs Number of communities", fontdict= {'size':15})
plt.xlabel('Number of communities')
plt.ylabel('Scores')
plt.legend()
```

ANNEXURE II: SNAPSHOTS OF THE OUTPUT

Dataset snapshot:

	source	target
0	17	1
1	72	1
2	3	2
3	19	2
4	20	2

Network Visualization:

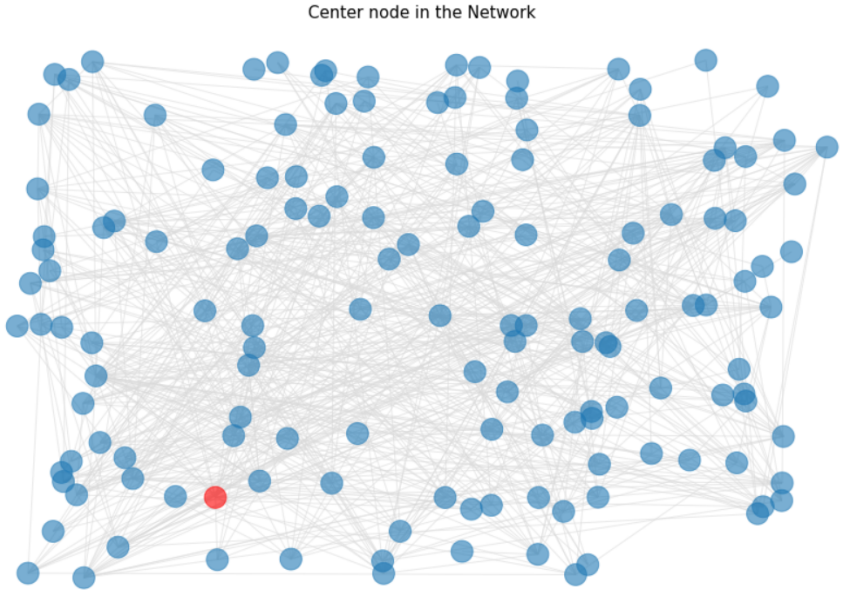


Basic metrics and finding if its a connected graph and if it is a bipartite graph

```
Max node degree (in + out) is 42
Max node degree (in + out) is 1
Diameter of the network is 8
Radius of the network is 4
Average distance in the network 2.967004826159756

Is bipartite? False
Is connected? True
```

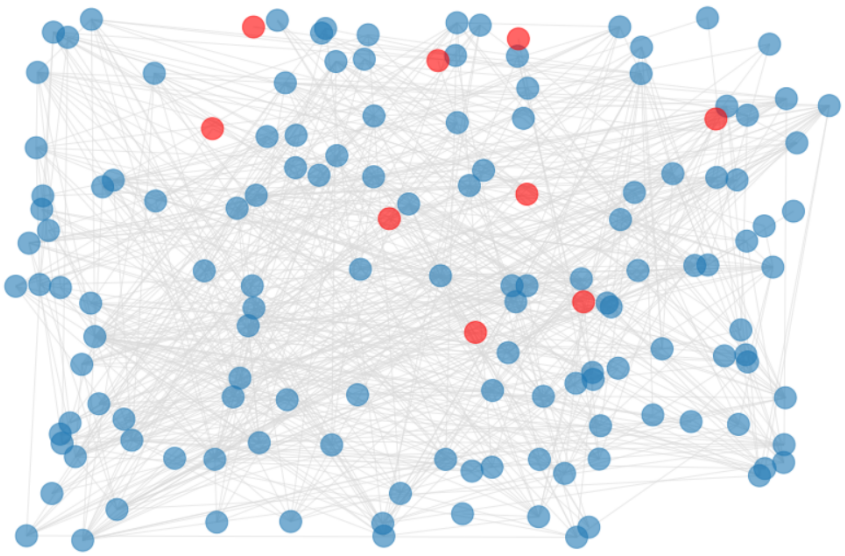
Central node in the network

```
Centers of the network:  [91]
Text(0.5, 1.0, 'Center node in the Network')
```



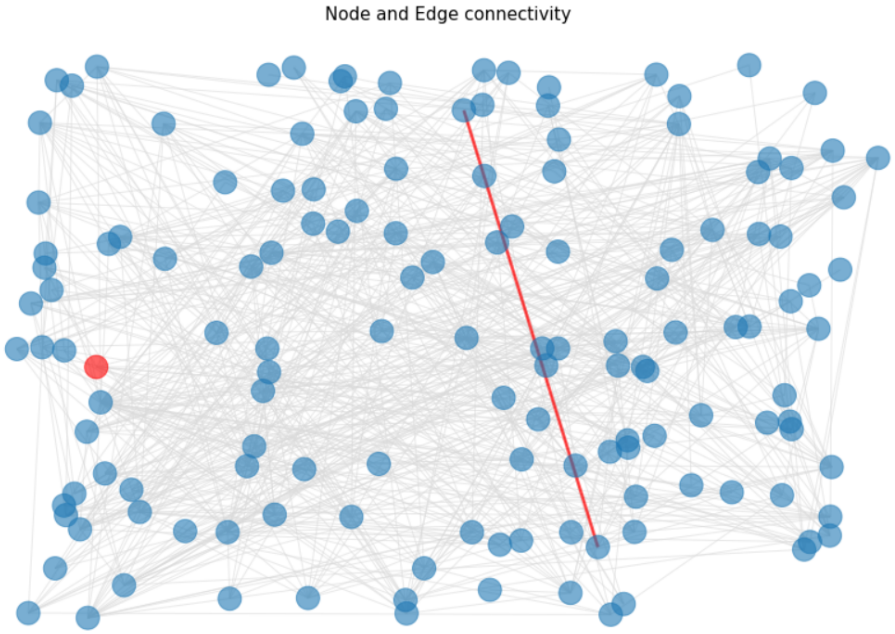
Periphery nodes in the network

```
Peripheries of the network:  [15, 42, 86, 84, 127, 113, 133, 92, 98]
Text(0.5, 1.0, 'Periphery nodes in Network')
```



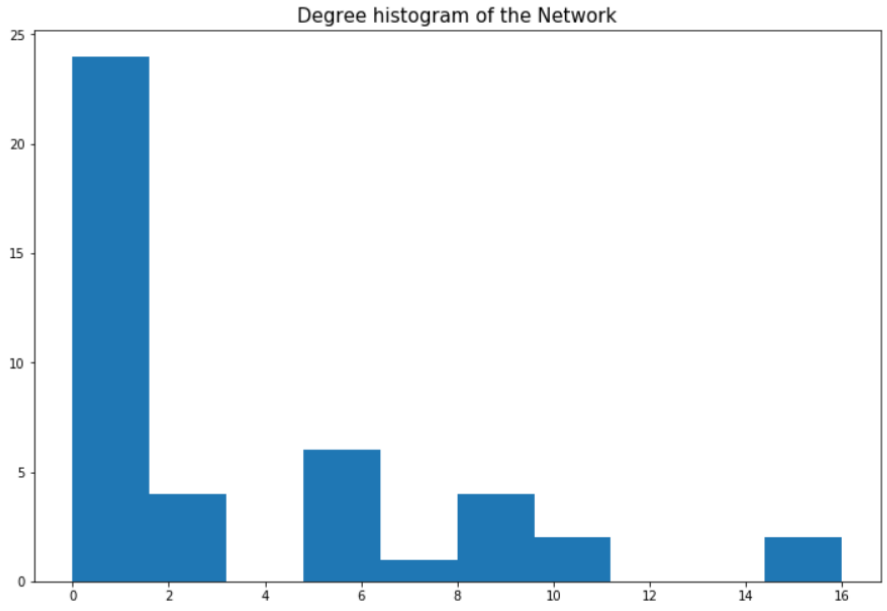
Disconnectivity of graph

```
Text(0.5, 1.0, 'Node and Edge connectivity')
```



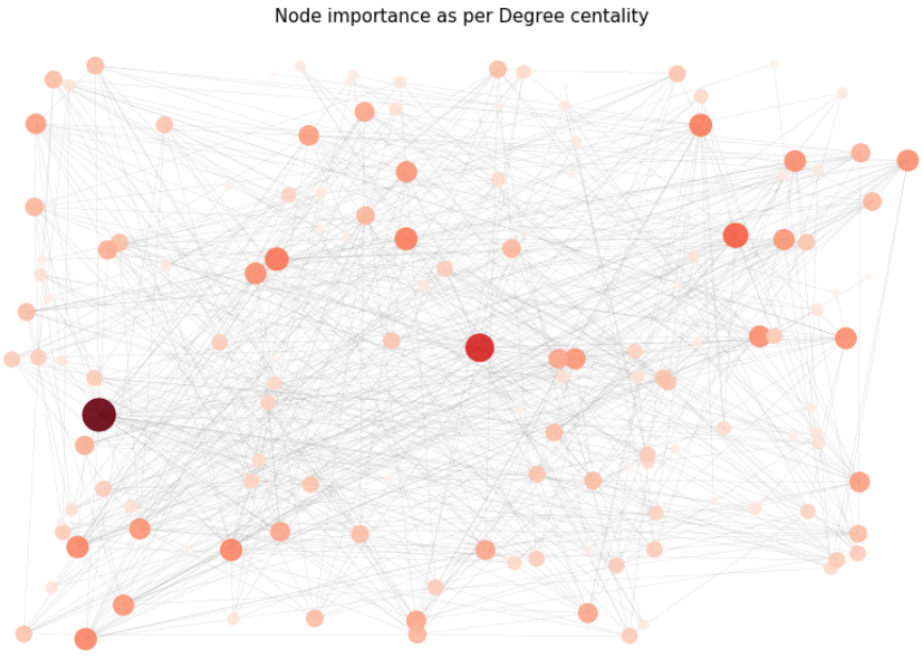
Degree Histogram of Network

```
Text(0.5, 1.0, 'Degree histogram of the Network')
```



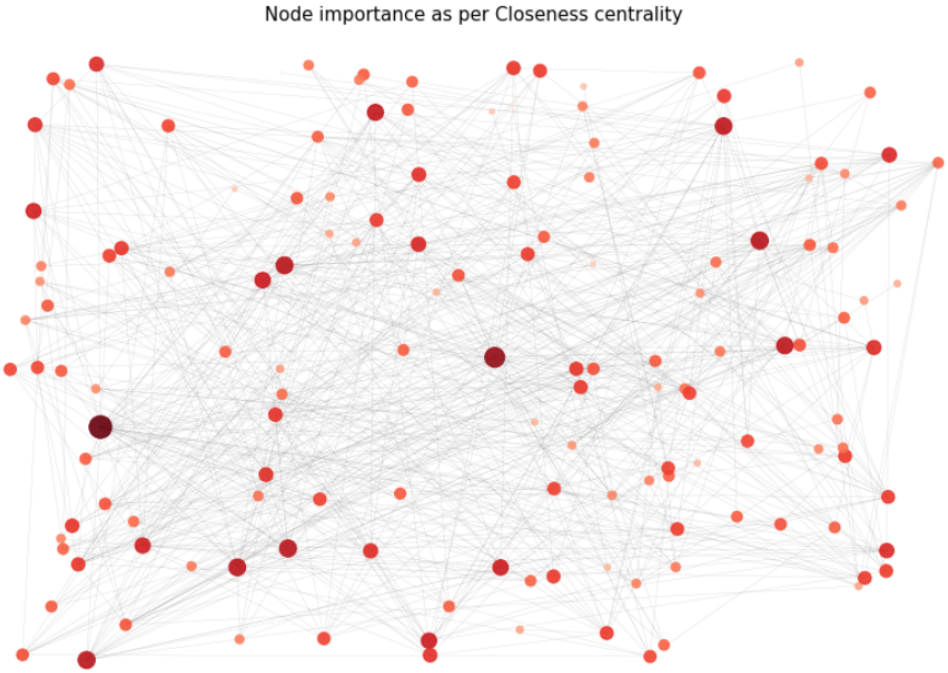
Degree Centrality

```
10 most important nodes as per Degree centrality
 [105, 17, 95, 48, 132, 43, 31, 74, 91, 72]
```



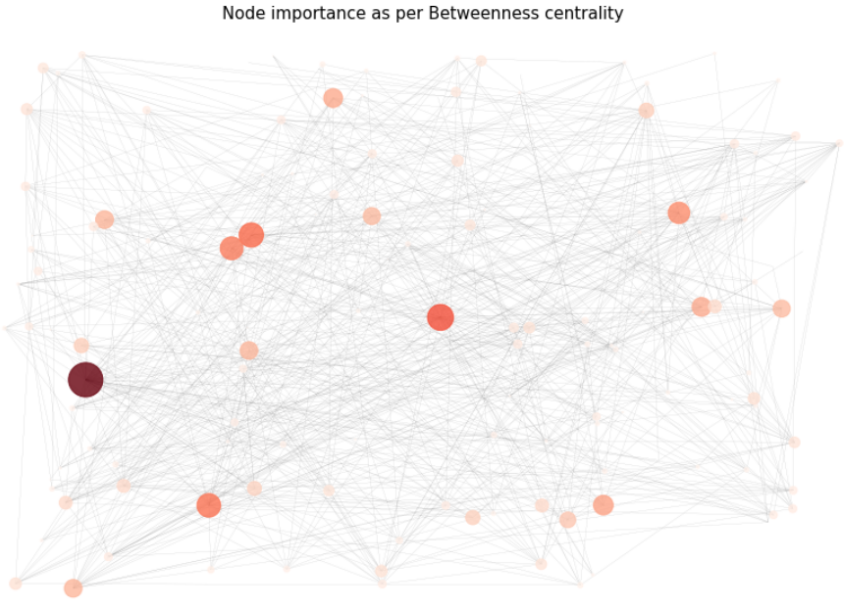
Closeness Centrality

```
10 most important nodes as per Closeness centrality
 [105, 17, 95, 74, 37, 48, 91, 43, 72, 22]
```



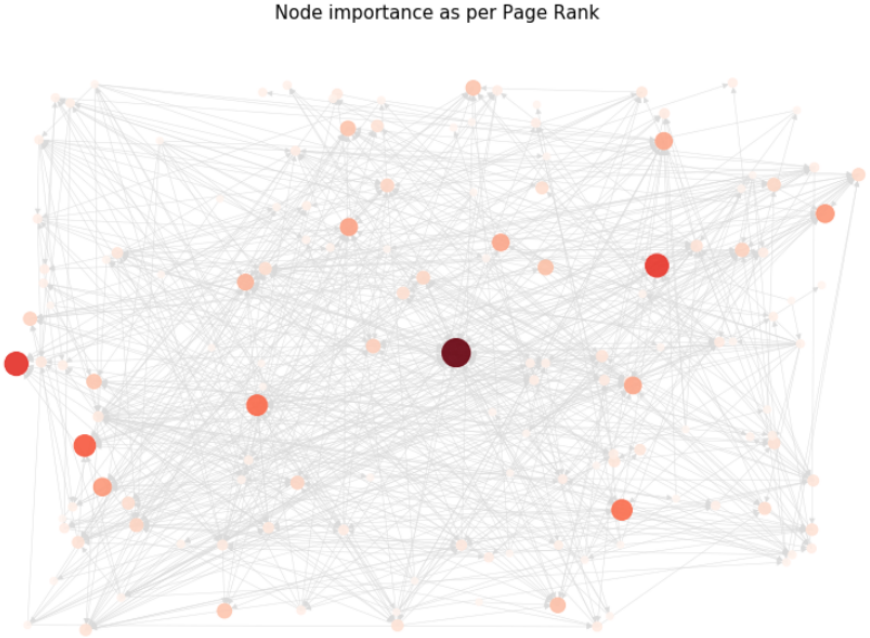
Betweeness Centrality

```
10 most important nodes as per Betweenness centrality
 [105, 17, 48, 91, 32, 95, 141, 22, 72, 51]
```



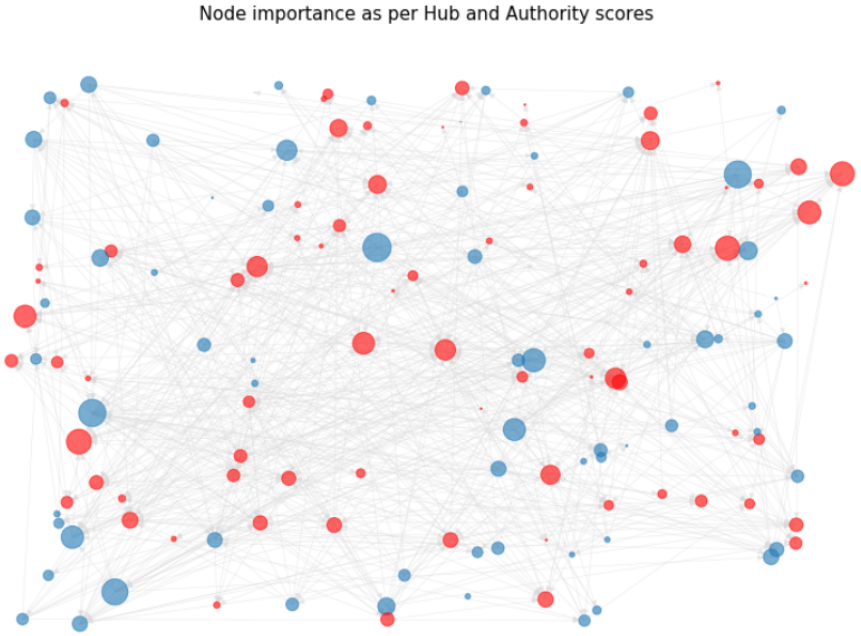
Page Rank

```
10 most important nodes as per Page Rank
 [17, 6, 7, 2, 10, 11, 3, 14, 9, 43]
```

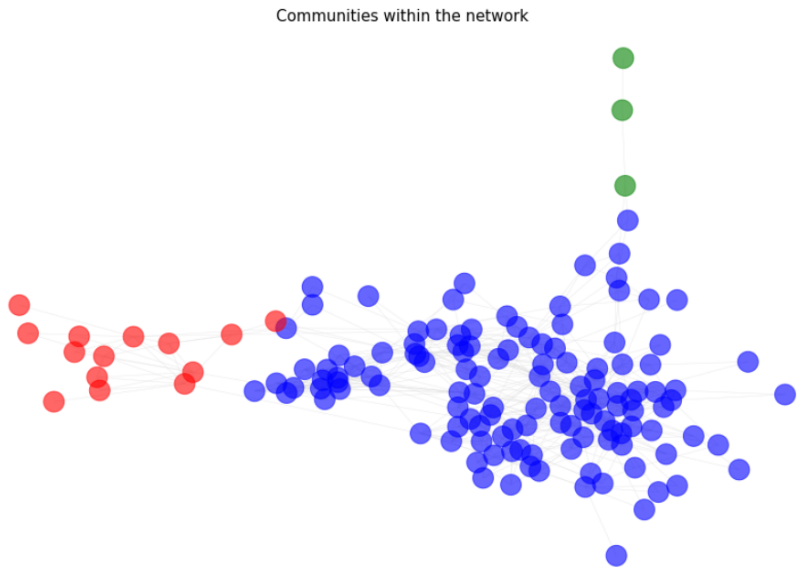


HITS Algorithm

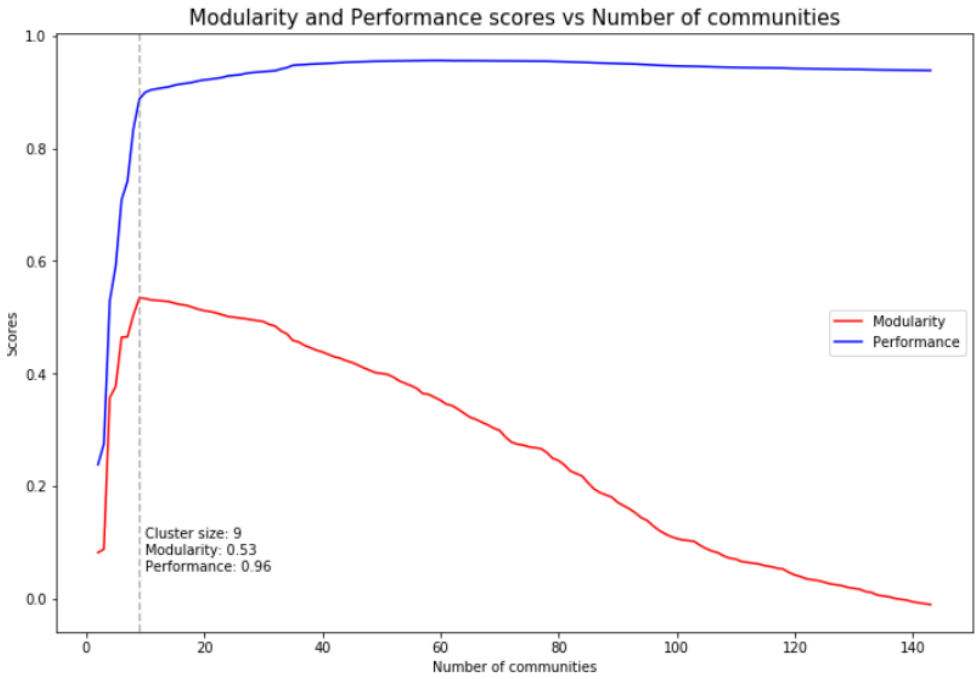
```
10 most important nodes as per Hub scores
 [132, 105, 136, 69, 67, 31, 71, 65, 48, 27]

10 most important nodes as per Authority scores
 [2, 19, 20, 3, 4, 18, 31, 17, 28, 48]
```



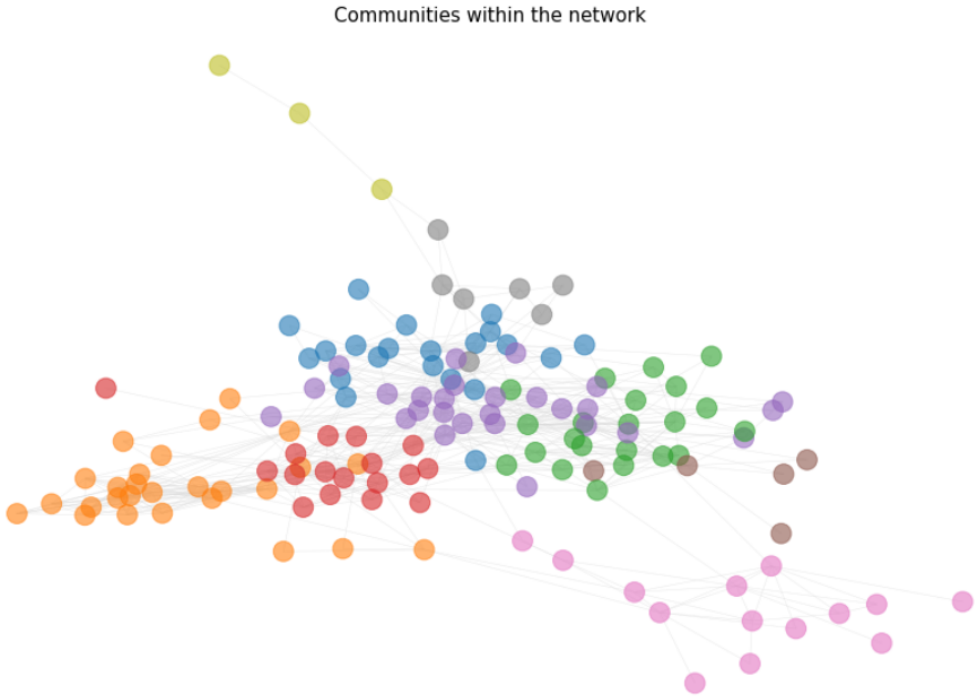
Girvan Newman Algorithm to split into 3 communities



Identifying optimal community split



Girvan Newman Algorithm to split into optimal number of communities



REFERENCES

1. https://towardsdatascience.com/deriving-patterns-of-fraud-from-the-enron-dataset-64cbceb65c36
2. https://www.kaggle.com/wcukierski/enron-email-dataset
3. https://www.tandfonline.com/doi/pdf/10.1080/10691898.2015.11889734
4. https://foreverdata.org/1009HOLD/Enron_Dataset_Report.pdf
5. https://networkx.org/documentation/stable/tutorial.html
6. https://www.geeksforgeeks.org/hyperlink-induced-topic-search-hits-algorithm-using-networxx-module-python/amp/
7. https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.link_analysis.pagerank_alg.pagerank.html
8. https://networkx.guide/algorithms/community-detection/girvan-newman/https://www.geeksforgeeks.org/networkx-python-software-package-study-complex-networks/
9. https://medium.com/swlh/a-tutorial-on-networkx-network-analysis-in-python-part-i-43c1d35830b6
10. https://medium.com/analytics-vidhya/a-gentle-introduction-to-networkx-with-python-21c29419d28a
11. http://pi.math.cornell.edu/~mec/Winter2009/RalucaRemus/Lecture4/lecture4.html
12. https://www.google.com/amp/s/www.geeksforgeeks.org/python-visualize-graphs-generated-in-networkx-using-matplotlib/amp/