M.KUMARASAMY
COLLEGE OF ENGINEERING
NAAC Accredited Autonomous Institution
Approved by AICTE & Affiliated to Anna University
ISO 9001:2015 & ISO 14001:2015 Certified Institution
Thalavapalayam, Karur – 639 113.

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

EXPERIENTIAL LEARNING-1

18AIE010T –

WEB & SOCIAL MEDIA MINING

TEAM MEMBERS:
- NIKITHA Y S (927621BAD035)
- NIVEDHA M (927621BAD036)
- VINOHARSITHA (927621BAD060)

SUBMITTED TO :

- **Dr.T SARAVANAN ,**
  Asst prof /AI

SUBMIT DATE : MARCH 04, 2024

FACEBOOK DATA MINING

INTRODUCTION:

Social network analysis provides invaluable insights into the dynamics of online interactions, making it a crucial area of study for data enthusiasts. Leveraging Facebook data for mining techniques allows learners to explore real-world datasets and gain practical experience in Python programming. This report aims to encourage learners to implement mining techniques on the Facebook dataset using Python, emphasizing hands-on coding exercises to deepen their understanding of social network analysis.

ABOUT THE DATASET:

This dataset consists of 'circles' (or friends lists') from Facebook. Facebook data was collected from survey participants using this Facebook app. The dataset includes node features (profiles), circles, and ego networks.

Facebook data has been anonymized by replacing the Facebook-internal ids for each user with a new value. Also, while feature vectors from this dataset have been provided, the interpretation of those features has been obscured. For instance, where the original dataset may have contained a feature "political=Democratic Party", the new data would simply contain "political anonymized feature 1". Thus, using the anonymized data it is possible to determine whether two users have the same political affiliations, but not what their individual political affiliations represent.

UNDERSTANDING FACEBOOK DATA:

The Facebook dataset encompasses various dimensions of social interactions, including user profiles, friendship connections, post interactions, and user-generated content. Each facet offers unique insights into user behavior, network structure, and content engagement. By analyzing this data, learners can uncover patterns, detect communities, and understand the dynamics of online social networks**.**

SELECTING MINING TECHNIQUES:

For the Facebook dataset, learners can explore a range of mining techniques tailored to social network analysis. These may include:

1)Community Detection: Identifying cohesive groups or communities within the network based on connectivity patterns.

2)Centrality Measures: Assessing the importance of nodes within the network using metrics like degree centrality, betweenness centrality, and eigenvector centrality.

3)Sentiment Analysis: Analyzing user-generated content to gauge sentiment and attitudes within the network.

These techniques offer valuable insights into network structure, node importance, and user behavior, enabling learners to extract meaningful information from the Facebook dataset.

IMPLEMENTATION:

Python, with its rich ecosystem of libraries, is an ideal choice for implementing mining techniques on the Facebook dataset. Learners can utilize libraries such as NetworkX for network analysis, Pandas for data manipulation, and Matplotlib for visualization. By writing Python code, learners can preprocess the dataset, apply mining algorithms, and visualize results in an interactive and intuitive manner.

Coding exercises form the core of implementing mining techniques in Python. Learners can engage in hands-on activities, including:

1)Loading and preprocessing the Facebook dataset using Pandas.

2)Constructing a social network graph using NetworkX.

3)Applying community detection algorithms to identify clusters within the network.

4)Calculating centrality measures to assess node importance and influence.

In [1]:
```
import networkx as nx
import matplotlib.pyplot as plt
```

In [2]:
```
G1 =nx.read_edgelist("facebook_combined.txt", create_using = nx.Graph(), nodetype=int)
G1
```

Out[2]:
```
<networkx.classes.graph.Graph at 0x1efa4e256a0>
```

In [3]:
```
print(nx.info(G1))
```

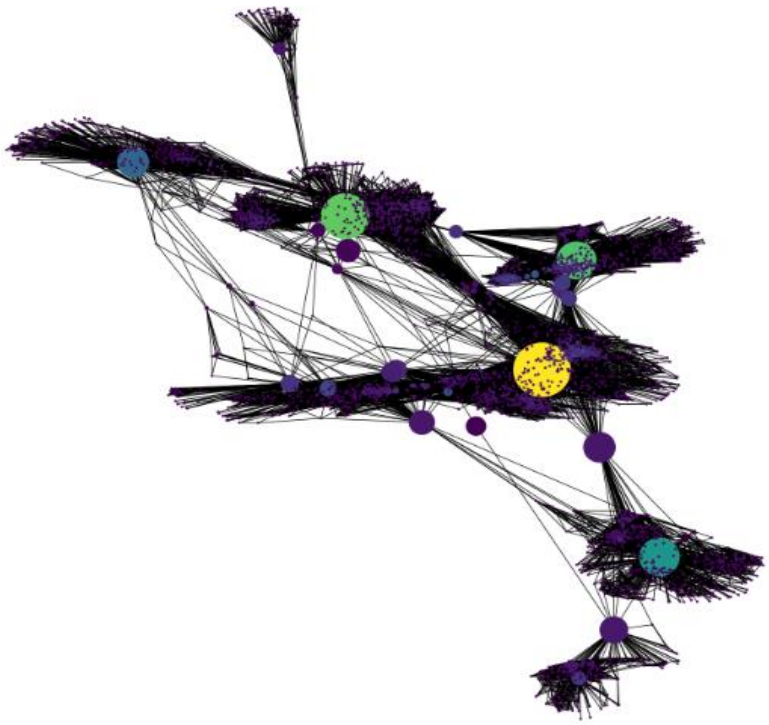
```
Graph with 4039 nodes and 88234 edges
```

```
C:\Users\Nikitha\AppData\Local\Temp\ipykernel_23408\2189145560.py:1: DeprecationWarning: info is deprecated and will be removed in version 3.0.

  print(nx.info(G1))
```

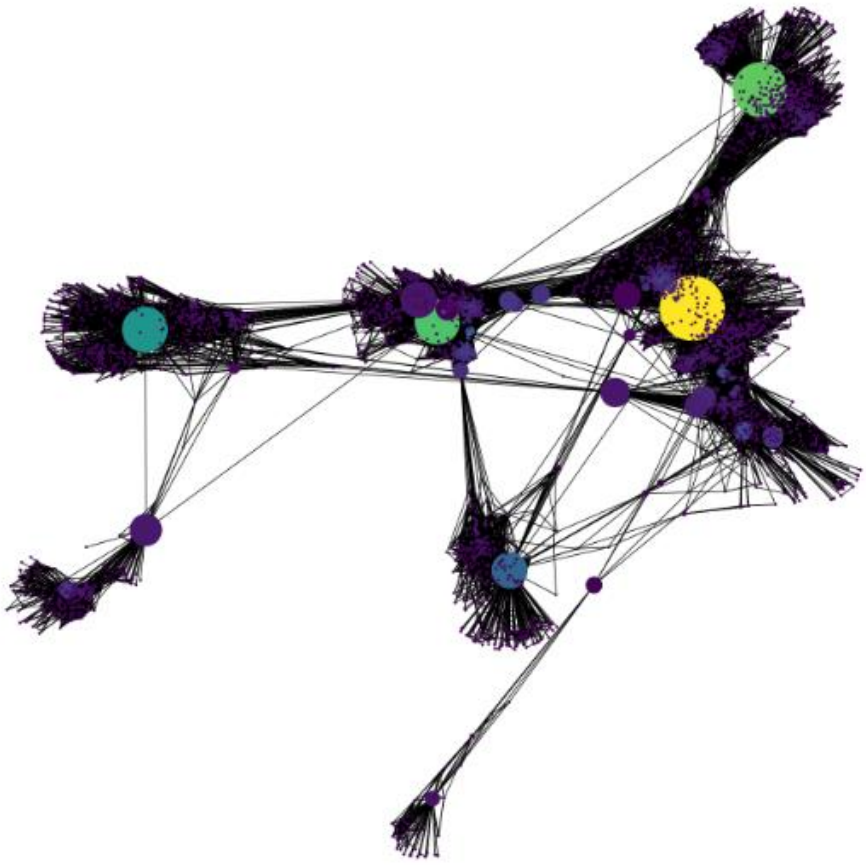
In [4]:
```
pos = nx.spring_layout(G1)
betCent = nx.betweenness_centrality(G1, normalized=True, endpoints=True)
node_color = [20000.0 * G1.degree(v) for v in G1]
node_size =  [v * 10000 for v in betCent.values()]
plt.figure(figsize=(20,20))
nx.draw_networkx(G1, pos=pos, with_labels=False,
                 node_color=node_color,
                 node_size=node_size )
plt.axis('off')
sorted(betCent, key=betCent.get, reverse=True)[:5]
```

Out[4]:
```
[107, 1684, 3437, 1912, 1085]
```



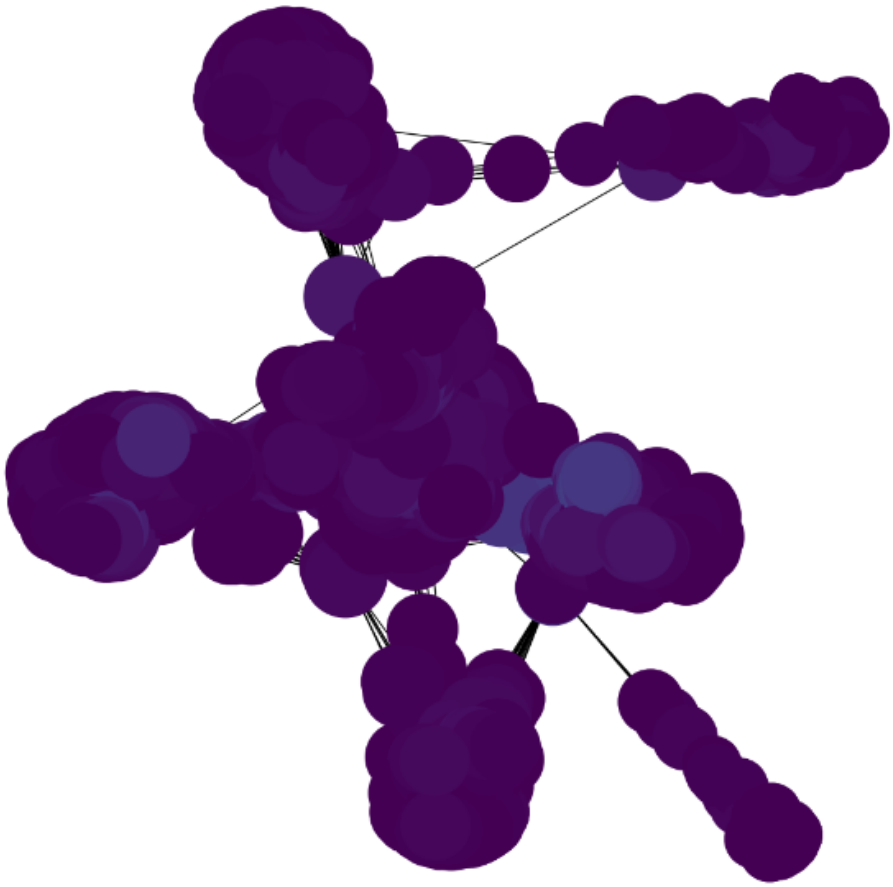
In [5]:
```
pos = nx.spring_layout(G1)
betCent = nx.betweenness_centrality(G1, normalized=True, endpoints=True)
node_color = [20000.0 * G1.degree(v) for v in G1]
node_size =  [v * 10000 for v in betCent.values()]
plt.figure(figsize=(20,20))
nx.draw_networkx(G1, pos=pos, with_labels=False,
                 node_color=node_color,
                 node_size=node_size )
plt.axis('off')
sorted(betCent, key=betCent.get, reverse=True)[:5]
```

Out[5]:
```
[107, 1684, 3437, 1912, 1085]
```



In [8]:
```
pos = nx.spring_layout(G1)
cloCent = nx.closeness_centrality(G1)
node_color = [20000.0 * G1.degree(v) for v in G1]
node_size =  [v * 10000 for v in cloCent.values()]
plt.figure(figsize=(13,13))
nx.draw_networkx(G1, pos=pos, with_labels=False,
                 node_color=node_color,
                 node_size=node_size )
plt.axis('off')
sorted(cloCent, key=cloCent.get, reverse=True)[:5]
```

Out[8]: [107, 58, 428, 563, 1684]



In [9]:
```
max(x for x,y in nx.degree(G1))
```

Out[9]:
```
4038
```

In [10]:
```
sources = [20,40,65,75]
targets = [650,802,920,1010]
for i in range(4):
    path = nx.shortest_path(G1,source=sources[i],target=targets[i])
    length = nx.shortest_path_length(G1,source=sources[i],target=targets[i],method='dijkstra')
    print("Shortest Path between Node ", str(sources[i])," ---> ", str(targets[i]), " is ",
          str(path), " ,Length = ", str(length))
```

```
Shortest Path between Node  20  --->  650  is  [20, 0, 34, 414, 650]  ,Length =  4
Shortest Path between Node  40  --->  802  is  [40, 0, 58, 1684, 860, 698, 686, 802]  ,Length =  7
Shortest Path between Node  65  --->  920  is  [65, 0, 107, 920]  ,Length =  3
Shortest Path between Node  75  --->  1010  is  [75, 0, 107, 1010]  ,Length =  3
```

In [11]:
```
neigh = [1,20,40,65,75,90,1000,]
for i in range(len(neigh)):
    all_neighbors = list(nx.classes.function.all_neighbors(G1,neigh[i]))
    print("All neighbors for Node ", str(neigh[i])," ---> ", str(all_neighbors))
```

```
All neighbors for Node  1  --->  [0, 48, 53, 54, 73, 88, 92, 119, 126, 133, 194, 236, 280, 299, 315, 322, 346]
All neighbors for Node  20  --->  [0, 2, 14, 41, 44, 111, 115, 149, 162, 214, 226, 312, 326, 333, 343]
All neighbors for Node  40  --->  [0, 21, 25, 26, 29, 56, 67, 72, 77, 113, 132, 133, 141, 142, 158, 169, 172, 199, 200, 203, 21
2, 213, 224, 231, 232, 239, 257, 258, 265, 271, 272, 274, 277, 280, 298, 304, 307, 315, 317, 322, 325, 329, 332, 334]
All neighbors for Node  65  --->  [0, 7, 13, 25, 82, 118, 203, 252, 261, 297, 314, 339]
All neighbors for Node  75  --->  [0, 9, 56, 67, 85, 170, 188, 200, 258, 272, 274, 304, 322, 323]
All neighbors for Node  90  --->  [0, 179]
All neighbors for Node  1000  --->  [107, 924, 974, 985, 1010, 1127, 1134, 1228, 1304, 1474, 1640, 1667, 1703, 1725, 1759, 184
0]
```

In [12]:
```
list(nx.selfloop_edges(G1, keys=True, data=True))
#None []
```

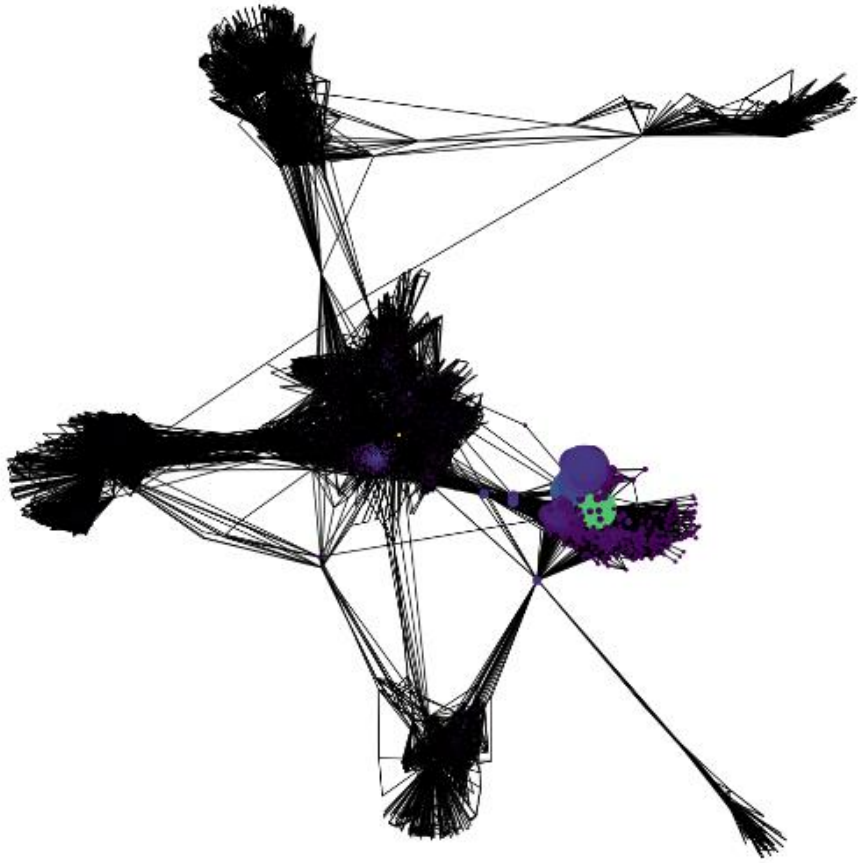
Out[12]:
```
[]
```

In [13]:
```
d = dict(G1.degree()).values()
from collections import Counter
Counter(d)
```

Out[13]:
```
Counter({347: 1,
         17: 76,
         10: 95,
         13: 79,
         6: 98,
         20: 63,
         8: 111,
         57: 23,
         1: 75,
         31: 38,
         15: 106,
         9: 100,
         16: 82,
         65: 20,
         11: 81,
         69: 14,
         68: 16,
         5: 93,
         23: 53,
```

In [14]:
```
#pos = nx.spring_layout(G1)
eigCent = nx.eigenvector_centrality(G1)
node_color = [20000.0 * G1.degree(v) for v in G1]
node_size =  [v * 10000 for v in eigCent.values()]
plt.figure(figsize=(15,15))
nx.draw_networkx(G1, pos=pos, with_labels=False,
                 node_color=node_color,
                 node_size=node_size )
plt.axis('off')
```

Out[14]:
```
(-0.9399058285355568,
 1.1285278859734535,
 -1.2047096189856528,
 1.154325037896633)
```



In [15]:
```
sorted(eigCent, key=eigCent.get, reverse=True)[:5]
```

Out[15]:
```
[1912, 2266, 2206, 2233, 2464]
```

CONCLUSION:

Encouraging learners to implement mining techniques on the Facebook dataset using Python facilitates hands-on learning and skill development. By leveraging Python's versatility and libraries like NetworkX, learners can explore the intricacies of social network analysis and gain insights into online interactions. Through coding exercises, learners not only deepen their understanding of mining techniques but also acquire valuable skills applicable to various data analysis tasks. Ultimately, harnessing social network analysis with Python empowers learners to navigate complex datasets and extract meaningful insights from online social networks.