Branch: 2IA

Project Title: Unveiling the Twittersphere: Community Detection Analysis.

Mohamed IFQIR

Yassine SEDJARI

Prof: LAZAAR MOHAMED

Outline:

I. Libraries Used:

Entrée [41]:

```
import pandas as pd
import numpy as np
import os
import glob
from sklearn.cluster import KMeans
from sklearn import metrics
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np
from scipy.sparse import lil_matrix
from sklearn.cluster import SpectralClustering
from sklearn.metrics import silhouette_score
import seaborn as sns
```

II. Materials and Methods:

Method 1: Edge-based Approach

Graph represantation of data

Entrée [6]:

```
directory = "twitter/"

# Construct the file pattern to match
pattern = os.path.join(directory, "*.featnames")

# Get a list of filenames that match the pattern
filenames = glob.glob(pattern)

nodeIds = []
for filename in filenames:
    starting_index = filename.find("/")+1
    ending_index = filename.find(".")
    nodeIds.append(int(filename[starting_index:ending_index]))
```

Entrée [7]:

```
# Step 1: Load the dataset
edges = []
with open("twitter_combined.txt", "r") as file:
    for line in file:
        source, target = line.strip().split()
        edges.append((int(source), int(target)))
```

Entrée [8]:

```
selected_edges = []
for edge in edges:
    if (edge[0] in nodeIds) and (edge[1] in nodeIds):
        selected_edges.append(edge)
selected_edges
```

```
 (355823615, 148519842),
 (160237722, 40981798),
 (170167167, 207594668),
 (248883350, 314316607),
 (18996905, 40981798),
 (265077741, 158419434),
 (148519842, 355823615),
 (307458983, 115221382),
 (43003845, 158419434),
 (206923844, 158419434),
 (166214735, 158419434),
 (158419434, 307458983),
 (17627996, 40981798),
 (187773078, 43003845),
 (93906304, 79797834),
 (79797834, 22879382),
 (37977732, 79797834),
 (37977732, 93906304),
 (93906304, 37977732),
```

Entrée [9]:

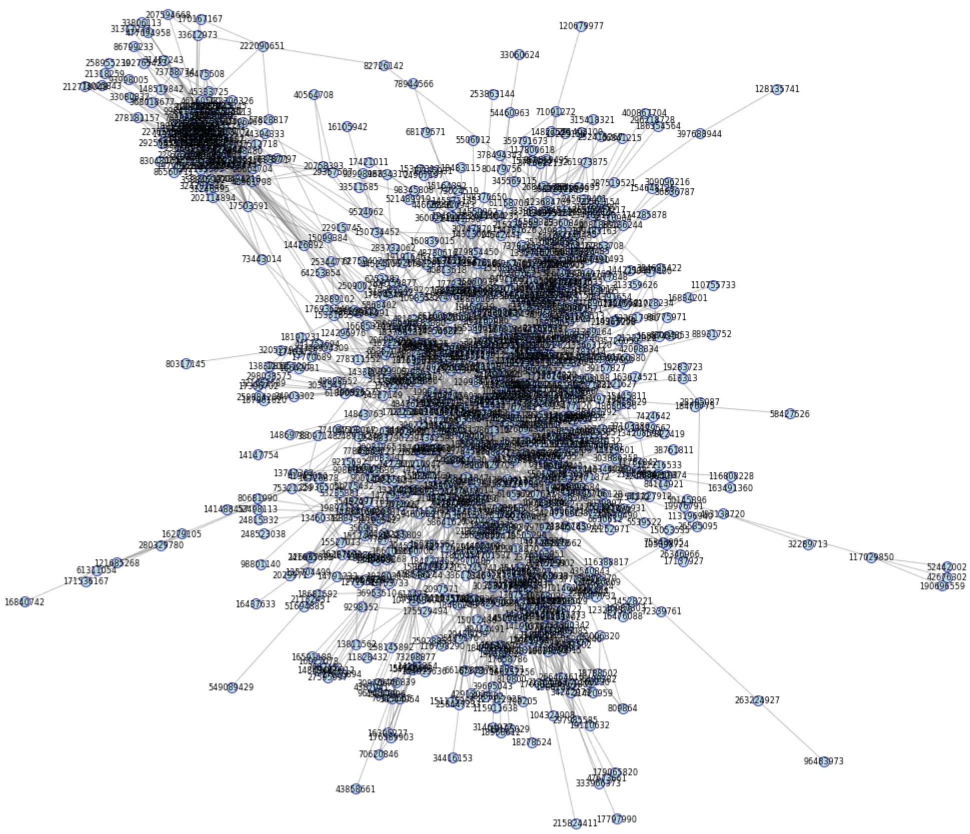
```
edges = selected_edges
```

Entrée [12]:

```
# Step 2: Create a graph representation
graph = nx.Graph()
graph.add_edges_from(edges)

# Step 3: Plot a subset of the graph
subset_nodes = list(graph.nodes)[:1000]  # Select the first 1000 noc

subset_graph = graph.subgraph(subset_nodes)

# Use Circular layout for better performance with large graphs
pos = nx.spring_layout(subset_graph)
```

Entrée [13]:
```
plt.figure(figsize=(20, 18))

# Draw nodes with custom style
nx.draw_networkx_nodes(subset_graph, pos, node_color='lightblue', nc

# Draw edges with custom style
nx.draw_networkx_edges(subset_graph, pos, width=1.0, alpha=0.5, edge

# Draw labels for nodes
nx.draw_networkx_labels(subset_graph, pos, font_size=8, font_color='

plt.axis('off')
plt.title('Subset of Graph')
plt.show()
```

Subset of Graph



Data Preprocessing: Adjacency Matrix

Entrée [20]:
```
# Get the number of node
nodes = set()

for edge in edges:
    nodes.add(edge[0])
    nodes.add(edge[1])
num_nodes = len(nodes)

# Create a dictionary to map node IDs to indices
node_to_index = {node: index for index, node in enumerate(nodes)}

# Create the adjacency matrix (sparse)
adjacency_matrix = lil_matrix((num_nodes, num_nodes), dtype=np.int8)
for edge in edges:
    source, target = edge
    source_index = node_to_index[source]
    target_index = node_to_index[target]
    adjacency_matrix[source_index, target_index] = 1

# Convert to a compressed sparse row (CSR) matrix for efficient comp
adjacency_matrix = adjacency_matrix.tocsr()
adjacency_matrix.shape
```

Out[20]:
```
(972, 972)
```

Modeling: Spectral Clustering

Entrée [24]:
```
# Create an instance of SpectralClustering
model = SpectralClustering(n_clusters=3, affinity='precomputed')

# Fit the model using the adjacency matrix
clusters = model.fit_predict(adjacency_matrix)
```

```
/home/yassine/anaconda3/lib/python3.9/site-packages/sklearn/manifol
d/_spectral_embedding.py:234: UserWarning: Array is not symmetric,
and will be converted to symmetric by average with its transpose.
  adjacency = check_symmetric(adjacency)
```

Metric Evaluation: Spectral Clustering

Entrée [25]:

```
# Calculate the Silhouette score
silhouette = silhouette_score(adjacency_matrix, clusters)

# Print the assigned cluster labels and Silhouette score
print("Cluster Labels:", clusters)
print("Silhouette Score:", silhouette)
```

```
Cluster Labels: [0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 2 0 0 0
0 2 0 0 0 0 0 2 0 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 2 2
 0 0 2 0 0 0 0 0 0 0 2 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 2
 0 2 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 2 2 0 2 0 0 0 0 0
0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 2 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 2 0 0 0 2 0 0 0 0
0 0 0 0
 0 0 2 0 0 2 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0
0 0 0 0
 0 0 0 0 2 2 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 2 2
2 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 2 0 0
0 0 0 0
 0 0 0 0 0 0 0 0 2 0 0 0 0 2 0 0 0 2 0 0 0 0 0 2 0 0 0 0 2 0 2 0 0
0 0 2 0
 2 2 2 0 0 0 2 2 0 0 0 0 0 0 2 1 2 0 0 0 2 0 2 0 0 0 0 2 0 2 0 0 0
0 0 0 0
 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 2 0 0 2 0 0 0 0 2 0 0 0
0 0 0 0
 0 0 0 0 0 0 0 2 0 0 0 0 0 0 2 2 0 0 0 0 0 0 0 0 0 2 0 0 0 0 2 2 2
2 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 2 0 0 1 0 0 0 0 0 0 0 2 0
0 0 0 0
 2 0 0 2 0 0 0 2 2 2 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 2 0 2 0 2
0 0 0 0
 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 2 0 0 0 0 0 2 2
0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 0 0 0 0
0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2
0 0 0 2
 0 0 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 0 0 0 0 0 0 0 0
0 0 2 0
 0 0 2 0 0 0 0 0 0 0 0 0 2 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 2 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 2 0 0 0 2 0 0 0 0 0 0 2 0 0 2
0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 2 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 2 0 0 0 0 2 0 0 0 0
2 0 0 0
 0 0 0 0 2 2 0 0 0 0 0 0 2 0 2 0 0 0 0 0 0 0 0 0 0 2 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 2 2 0 0 0 0 0 0 0 0 0 2
2 2 0 0
 0 2 0 0 0 0 0 0 0 0]
Silhouette Score: -0.18874041008983813
```

Method 2: Feature-based Approach

Tabular represantation of data

Entrée [26]:

```
hm = {}
for nodeId in nodeIds:
    file_path_featnames = "twitter/"+str(nodeId)+".featnames"
    file_path_egofeat = "twitter/"+str(nodeId)+".egofeat"

    # Open the file in read mode
    with open(file_path_egofeat, "r") as file:
        # Read the entire contents of the file
        egofeat = file.read().split()

    with open(file_path_featnames, "r") as file:
        # Read the entire contents of the file
        featnames_raw = file.readlines()

    index_ones = [ index for index,char in enumerate(egofeat) if cha

    featurenames1=[]
    for line in featnames_raw:
        starting_index = line.find(" ")
        ending_index = line.find('\n')
        featurenames1.append(line[starting_index+1:ending_index])

    for index in index_ones:
        featurename = featurenames1[index]
        if featurename not in hm:
            hm[featurename]=[]
            hm[featurename].append(nodeId)
        else:
            hm[featurename].append(nodeId)
hm
```

```
 '@astowellcom': [207594668],
 '@emmastonebr': [207594668],
 '@helpmovie': [207594668],
 '@igorbenhuy': [207594668],
 '@jessicabielorg': [207594668],
 '@katyperry:': [207594668, 31457243, 358845982, 789071],
 '@octaviaspencer:': [207594668],
 '#BrothersToTheEnd': [111374622],
 '#drunktweet': [111374622],
 '#gears3': [111374622, 289738351],
 '#gearsweekend': [111374622],
 '#nerdgasm': [111374622],
 '@FINALLEVEL': [111374622, 382110320, 129093262],
 '@GearsViking': [111374622, 289738351, 5774432, 155976326, 5368
5618],
 '@Kaylila': [111374622, 306445007],
 '@MissChelseaRene': [111374622],
 '@NastyShottyChic': [111374622],
 '@O5C4RM1KE': [111374622, 163629705, 105150583, 53685618],
 '@iCryptik': [111374622],
```

Entrée [27]:
```
df = pd.DataFrame(columns = ["nodeId"]+list(hm.keys()))
```

Entrée [28]:
```
df["nodeId"]=nodeIds
df = df.fillna(0)
for item,value in hm.items():
    featurename = item
    nodes = value
    for node in nodes:
        df.loc[ df["nodeId"]==node ,featurename] = 1
```

Entrée [29]:
```
df
```

Out[29]:

	nodeId	#OCTAVIA	#THEHELP	#ff	@BAFTA	@FuckYesEmma	@JUDAOcombr	@a
0	207594668	1	1	1	1	1	1	
1	111374622	0	0	0	0	0	0	
2	96483973	0	0	0	0	0	0	
3	7875912	0	0	0	0	0	0	
4	14147754	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	
968	14528221	0	0	1	0	0	0	
969	14840869	0	0	0	0	0	0	
970	82726142	0	0	0	0	0	0	
971	255790981	0	0	0	0	0	0	
972	36618690	0	0	0	0	0	0	

973 rows × 24246 columns

Entrée [30]:

```python
# Calculate degree centrality
degree_centrality = nx.degree_centrality(graph)
print("Degree Centrality:", degree_centrality)

# Calculate closeness centrality
closeness_centrality = nx.closeness_centrality(graph)
print("Closeness Centrality:", closeness_centrality)

# Calculate betweenness centrality
betweenness_centrality = nx.betweenness_centrality(graph)
print("Betweenness Centrality:", betweenness_centrality)
```

```
6261586, 85432934: 0.016477857878475798, 19563357: 0.01338825952
6261586, 18895362: 0.015447991761071O6, 23742633: 0.010298661174
047374, 7890392: 0.01544799176107106, 14505838: 0.01956745623069
001, 23759573: 0.008238928939237899, 20495756: 0.007209062821833
161, 54178513: 0.006179196704428424, 307478701: 0.00411946446961
8949, 276843589: 0.007209062821833161, 61311054: 0.0051493305870
23687, 16279105: 0.004119464469618949, 10146102: 0.0072090628218
33161, 16987303: 0.0020597322348094747, 17045060: 0.011328527291
45211, 21364753: 0.005149330587023687, 14199378: 0.0205973223480
9475, 317313520: 0.01544799176107106, 17600223: 0.00617919670442
8424, 5747502: 0.009268795056642637, 18119683: 0.007209062821833
161, 278311152: 0.004119464469618949, 38108292: 0.01029866117404
7374, 926981: 0.007209062821833161, 83883736: 0.0236869207003089
6, 7888452: 0.013388259526261586, 241635675: 0.00514933058702368
7, 4258591: 0.007209062821833161, 5539522: 0.003089598352214212,
9254272: 0.004119464469618949, 17787399: 0.007209062821833161, 1
8534908: 0.005149330587023687, 734493: 0.021627188465499485, 213
91704: 0.008238928939237899, 249829509: 0.009268795056642637, 34
5569115: 0.005149330587023687, 24542441: 0.014418125643666322, 3
```

Entrée [31]:

```python
df["Degree Centrality"]=np.nan
df["Closeness Centrality"]=np.nan
df["Betweenness Centrality"]=np.nan
```

```
/tmp/ipykernel_3560/2905337359.py:1: PerformanceWarning: DataFrame
is highly fragmented.  This is usually the result of calling `fram
e.insert` many times, which has poor performance.  Consider joining
all columns at once using pd.concat(axis=1) instead. To get a de-fr
agmented frame, use `newframe = frame.copy()`
  df["Degree Centrality"]=np.nan
/tmp/ipykernel_3560/2905337359.py:2: PerformanceWarning: DataFrame
is highly fragmented.  This is usually the result of calling `fram
e.insert` many times, which has poor performance.  Consider joining
all columns at once using pd.concat(axis=1) instead. To get a de-fr
agmented frame, use `newframe = frame.copy()`
  df["Closeness Centrality"]=np.nan
/tmp/ipykernel_3560/2905337359.py:3: PerformanceWarning: DataFrame
is highly fragmented.  This is usually the result of calling `fram
e.insert` many times, which has poor performance.  Consider joining
all columns at once using pd.concat(axis=1) instead. To get a de-fr
agmented frame, use `newframe = frame.copy()`
  df["Betweenness Centrality"]=np.nan
```

Entrée [32]:
```
for item,value in degree_centrality.items():
    df.loc[df["nodeId"]==item, "Degree Centrality"]=value
for item,value in closeness_centrality.items():
    df.loc[df["nodeId"]==item, "Closeness Centrality"]=value
for item,value in betweenness_centrality.items():
    df.loc[df["nodeId"]==item, "Betweenness Centrality"]=value
```

Entrée [33]:
```
df.dropna(inplace=True)
df.to_csv("dataset.csv",index=False)
```

Clustering and Evaluatoin:

Entrée [34]:
```
def cluster_and_evaluate(model_type, num_clusters, data, **kwargs):
    """
    Cluster the data using the specified model and evaluate the clus

    Args:
        model_type (str): Type of clustering model to use. Valid opt
        num_clusters (int): Number of clusters.
        data (numpy.ndarray or pandas.DataFrame): Input data to be c
        **kwargs: Additional keyword arguments specific to the chose

    Returns:
        tuple: A tuple containing the following elements:
            - labels (numpy.ndarray): Cluster labels assigned to eac
            - evaluation_metrics (dict): Dictionary of evaluation me
    """
    if model_type == "k-means":
        model = KMeans(n_clusters=num_clusters, **kwargs)
    elif model_type == "hierarchical":
        model = AgglomerativeClustering(n_clusters=num_clusters, **k
    elif model_type == "spectral":
        model = SpectralClustering(n_clusters=num_clusters, **kwargs
    else:
        raise ValueError("Invalid model type. Valid options: 'k-mear

    # Fit the model and obtain the predicted labels
    labels = model.fit_predict(data)

    # Evaluate the clustering results
    evaluation_metrics = {
        "Silhouette Score": metrics.silhouette_score(data, labels),
    }

    return labels, evaluation_metrics
```

K-MEANS

Entrée [35]:

```
data=df
# Cluster the data using k-means with 3 clusters
labels, metricss = cluster_and_evaluate("k-means", 3, data, random_s

# Print the labels and evaluation metrics
print("Cluster Labels:", labels)
print("Evaluation Metrics:")
for metric, value in metricss.items():
    print(metric + ":", value)
```

```
Cluster Labels: [2 2 1 1 1 2 1 1 1 1 2 2 1 2 1 0 2 1 1 2 1 2 1 1
1 1 1 1 1 2 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 2 1 2 1 2 1 2 1 1 0 0 0 0 1 1 2 1 1 1 2 2 1 1 0
1 0 0 1 1
 1 1 1 1 1 1 0 2 0 2 1 1 1 1 0 1 1 1 2 2 1 1 0 1 2 2 2 1 1 1 1 1
1 0 1 1 2
 1 1 1 2 2 1 1 0 1 1 1 1 0 2 2 1 1 2 1 1 2 0 1 2 1 0 1 2 1 1 2 1
1 1 0 1 1
 1 1 1 2 2 1 1 1 2 1 1 1 1 1 1 2 1 1 1 2 1 1 1 1 1 2 1 1 1 2 1 1
1 1 1 1 1
 1 1 1 1 1 0 1 1 1 2 1 1 2 0 1 2 1 1 1 1 1 2 1 2 1 1 1 1 1 1 1 1
1 1 1 1 1
 1 1 2 2 1 1 1 1 1 1 1 1 0 1 1 2 1 2 2 2 1 2 1 1 1 0 1 1 2 1 1 2
2 0 1 2 0
 1 1 1 2 1 2 0 1 2 1 1 0 1 0 2 1 1 1 2 1 2 2 1 1 1 1 2 1 0 0 1 1
0 1 1 1 1
 2 1 1 1 1 1 0 1 1 2 1 1 1 2 2 1 1 1 1 2 1 2 1 1 2 1 1 0 1 2 2 1
1 1 1 1 0
 0 1 1 2 1 2 1 1 1 2 2 1 0 1 1 2 2 1 1 1 1 1 1 1 1 2 1 1 1 1 1 2
1 1 2 2 2
```

Hierarchical Clustering

Entrée [ ]:

```
data=df
# Cluster the data using k-means with 3 clusters
labels1, metricss1 = cluster_and_evaluate("hierarchical", 3, data)

# Print the labels and evaluation metrics
print("Cluster Labels:", labels1)
print("Evaluation Metrics:")
for metric, value in metricss1.items():
    print(metric + ":", value)
```

Spectral Clustering

Entrée [36]:

```
data=df
# Cluster the data using k-means with 3 clusters
labels2, metricss2 = cluster_and_evaluate("spectral", 3, data, rando

# Print the labels and evaluation metrics
print("Cluster Labels:", labels2)
print("Evaluation Metrics:")
for metric, value in metricss2.items():
    print(metric + ":", value)
```

```
/home/yassine/anaconda3/lib/python3.9/site-packages/sklearn/manifol
d/_spectral_embedding.py:260: UserWarning: Graph is not fully conne
cted, spectral embedding may not work as expected.
  warnings.warn(
```

```
Cluster Labels: [0 0 0 0 1 0 2 2 0 0 2 1 0 2 1 1 0 0 0 2 2 1 0 1 0
0 1 2 2 2 1 1 2 0 2 0 0
 1 0 2 0 0 1 0 0 2 0 0 2 0 1 2 1 0 2 1 2 1 2 2 2 2 2 0 1 0 1 2 0 0
2 1 0 2
 0 2 1 0 1 2 1 2 2 2 2 0 0 0 0 1 2 1 2 1 1 2 1 0 2 0 1 2 0 0 1 2 2
2 1 1 0
 1 2 0 0 0 2 0 1 1 2 1 0 1 0 2 1 0 0 0 1 0 0 2 1 1 0 2 1 1 0 0 2 1
0 1 0 2
 1 2 0 0 2 0 0 0 0 0 0 1 0 1 2 1 0 2 2 1 1 0 1 1 2 2 0 0 2 2 2 0 0
2 1 0 2
 2 0 0 2 1 1 0 2 1 0 2 2 0 1 2 0 0 2 0 0 2 1 0 1 1 0 2 1 2 0 2 1 2
1 0 1 0
 0 2 1 0 2 2 1 1 1 0 2 2 1 2 0 2 0 0 1 2 0 0 0 0 2 2 1 2 1 1 2 2 2
0 2 2 0
 1 0 2 0 2 2 0 2 1 0 2 1 2 0 2 0 2 2 1 1 1 0 1 0 2 2 1 2 2 2 0 2 2
1 0 2 2
 1 2 2 2 0 2 2 1 2 1 0 1 0 1 2 0 2 0 1 1 1 0 1 1 2 0 0 0 1 2 0 1 0
2 2 1 1
 1 1 2 2 2 0 1 0 0 1 0 2 0 1 0 2 1 1 0 0 1 2 1 1 0 2 2 2 2 1 0 2 1
2 1 0 2
 2 2 2 2 2 2 0 0 2 0 1 1 1 2 1 2 0 2 2 0 0 0 0 0 1 2 1 2 2 0 0 0 0
1 0 1 0
 2 1 1 0 1 2 1 0 2 0 0 2 0 2 0 1 2 1 2 1 1 0 2 2 1 1 2 2 2 0 0 0 2
1 0 0 2
 1 2 0 2 2 1 2 0 1 0 1 0 1 2 1 1 1 0 2 0 1 2 0 1 2 0 2 0 0 2 1 0 0
0 0 0 0
 1 0 2 0 2 2 1 2 1 0 2 0 1 0 1 1 0 2 1 0 1 1 0 1 1 2 1 0 1 0 2 0 2
0 2 2 0
 2 2 2 0 1 2 2 2 0 1 1 2 2 1 0 1 1 2 0 2 0 0 0 0 2 2 0 0 2 1 0 0 0
0 0 0 2
 0 1 2 1 1 0 0 2 2 2 2 0 0 0 0 1 2 2 2 1 1 2 1 2 0 2 0 1 2 1 2 2 1
0 1 0 0
 2 0 1 2 2 2 0 2 0 2 0 0 2 2 1 2 2 0 0 1 2 1 2 2 0 2 0 2 0 0 2 2 1
0 0 2 1
 0 1 2 1 0 0 0 0 0 0 0 2 1 1 1 2 2 2 1 2 0 1 0 0 1 0 2 2 0 1 0 1 1
2 2 0 2
 0 2 1 1 0 1 0 0 1 0 2 2 2 2 2 1 2 0 2 1 0 2 0 0 2 0 0 1 1 0 0 1 0
0 0 0 1
 0 1 1 0 2 0 2 0 1 0 0 0 1 2 2 2 0 0 1 0 1 2 0 0 2 1 0 2 0 0 2 0 1
0 1 1 1
 1 1 2 2 0 1 2 0 2 2 1 2 0 0 2 1 2 0 2 0 2 1 1 2 1 2 0 2 0 1 1 0 1
2 0 2 0
 1 2 1 2 1 1 2 0 0 1 2 2 0 0 2 1 1 2 1 0 1 0 1 2 0 0 2 2 2 1 1 0 2
1 1 1 1
 1 0 0 2 0 0 2 2 1 1 2 2 2 0 2 0 0 1 2 1 2 1 0 0 2 1 1 2 2 2 2 2 0
0 0 1 2
 0 2 2 1 0 1 0 0 2 1 0 0 2 0 2 0 0 2 2 1 0 1 2 1 0 1 0 0 1 1 0 2 0
1 0 2 2
 0 2 2 0 1 1 1 1 0 1 0 2 2 1 0 0 0 2 1 0 0 0 0 2 2 2 2 1 1 1 1 0 2
0 2 2 1
 0 0 0 2 0 2 2 1 0 1 0 1 1 1 1 0 2 0 1 1 1 1 0 0 1 1 2 2 1 1 0 2 1
0 1 0 0
 0 2 2 0 0 0 0 0 1 0]
Evaluation Metrics:
Silhouette Score: -0.034292667509396
```

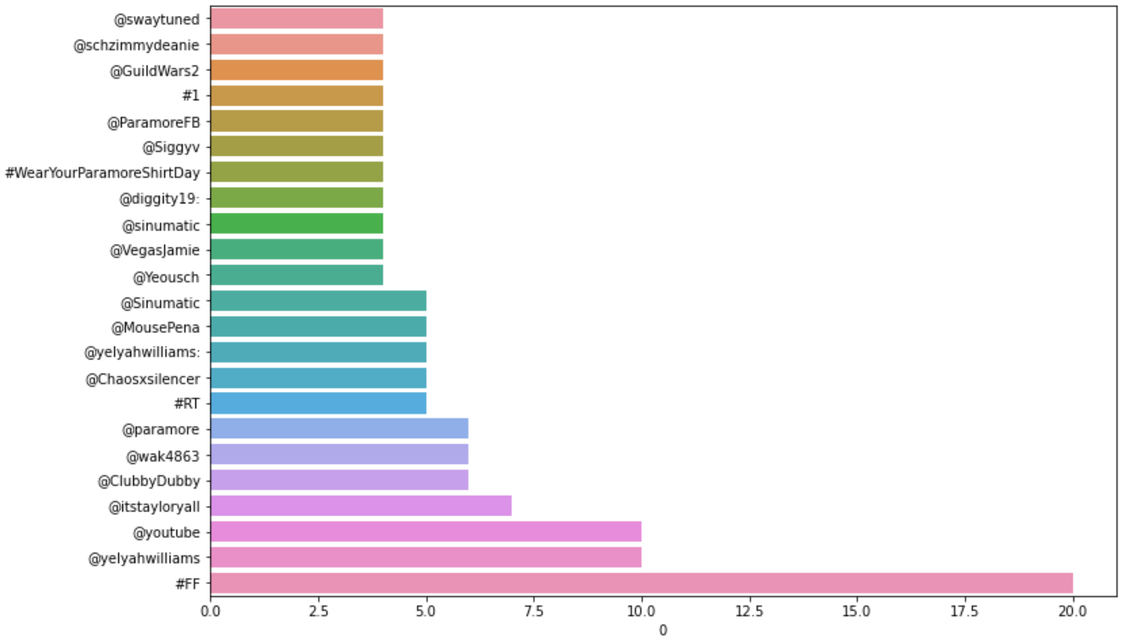
Labeling

Entrée [45]:
```
#taking labels of the best performing model: KMEANS
df["class"] = labels
df0 = df.loc[df["class"]==0]
df1 = df.loc[df["class"]==1]
df2 = df.loc[df["class"]==2]
```

Music Enthusiasts Cluster

Entrée [42]:
```
df_0_sum = ((df0.sum().sort_values()/df0.shape[0])*100).astype(int).
n0 = df_0_sum.shape[0]
df_0_sum.loc[(20>df_0_sum[0]) & (df_0_sum[0]>2),]
plt.figure(figsize=(12,8))
sns.barplot(y=df_0_sum.loc[(22>df_0_sum[0]) & (df_0_sum[0]>3),].inde
#Music
```

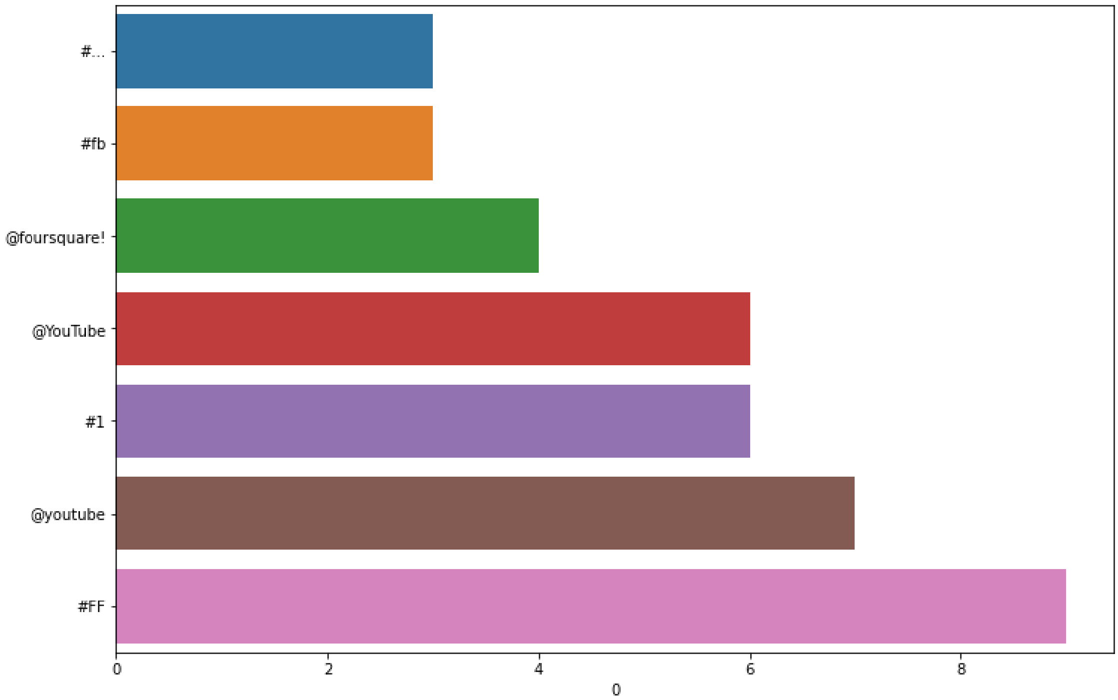
Out[42]: <AxesSubplot:xlabel='0'>



Social Media Enthusiasts Cluster

Entrée [43]:
```
df_1_sum = ((df1.sum().sort_values()/df1.shape[0])*100).astype(int).
n1 = df_1_sum.shape[0]
serie1 = df_1_sum.loc[(20>df_1_sum[0]) & (df_1_sum[0]>2),]
plt.figure(figsize=(12,8))
sns.barplot(
    y=serie1.index,
    x=serie1[0], orient='h')

#Social Media
```

Out[43]: <AxesSubplot:xlabel='0'>



Gaming Enthusiasts Cluster

Entrée [44]:
```
df_2_sum = ((df2.sum().sort_values()/df2.shape[0])*100).astype(int).
n2 = df_2_sum.shape[0]
serie2 = df_2_sum.loc[(23>df_2_sum[0]) & (df_2_sum[0]>2),]
plt.figure(figsize=(12,8))
sns.barplot(
    y=serie2.index,
    x=serie2[0], orient='h')

#Gaming
```

Out[44]: <AxesSubplot:xlabel='0'>