Objective:

This assignment aims to enhance your practical expertise in Python programming through the application of class-learned concepts to generate graphical representations. These representations aid in identifying critical edges and nodes. The program's specific tasks include computing cluster coefficients and neighborhood overlaps within an input graph. Furthermore, the graphs will be partitioned to emphasize primary groups and store it in a new graph. Mastery of these skills will not only reinforce your programming capabilities, essential for upcoming assignments but also heighten your proficiency and competitiveness in the dynamic field of Computer Science.

Requirement

The assignment requires that you create the Python program that runs in a terminal. The program accepts optional parameters. The syntax to run the program is shown in the following line:

```
python ./graph_analysis.py graph_file.gml --components n --plot
[C|N|P] --verify_homophily --verify_balanced_graph --output
out_graph_file.gml
```

Description of Parameters

This is the command to execute the Python script graph_analysis.py located in the current directory and reads the file graph_file.gml. Here, graph_file.gml is the file that will be used for the analysis and the format is Graph Modelling Language (.gml), which describes the graph's structure with attributes. The program should read the attributes of the nodes and edges in the file.

```
--components n
```

Specifies that the graph should be partitioned into n components. This divides the graph into n subgraphs or clusters.

```
--plot [C|N|P]
```

Determines how the graph should be plotted.

C: If this option is selected, the script will plot the graph highlighting clustering coefficients (which measure how nodes are tightly grouped).

The Cluster coefficient is proportional to its size. Let **cluster_min**, and **cluster_max** be the min and max cluster coefficients, and let **cv** be the clustering coefficient of node **v** and **pv = (cv-**

cluster_min)/(cluster_max - cluster_min) of node **v**. The siz **v** is proportional to **pv**. Let **max_pixel** and **min_pixel** be the minimum and maximum pixel sizes of the nodes. Therefore, the node **v** will have size **min_pixel + pv(max_pixel -min_pixel).**

The color of v is proportional to the degree. Similar as before, let **dv** be the degree of node **v,** **max** be the maximum degree in the graph. Let s**v = dv/max** be the size of node **v**. The RGB color of **v** is set to **(254*sv, 0, 254)** so the node with the lowest degree is Blue and the node with the highest degree is Magenta.

N: If this option is selected, the script will plot the graph highlighting neighborhood overlap (which measures how much overlap there is between neighborhoods of adjacent nodes). Similar to the clustering coefficient.

P: If this option is selected, the script will color the node according to the attribute if it is assigned, or a default color if not.

```
--verify_homophily
```

Tests for homophily in the graph based on the assigned node colors using the Student t-test. Homophily measures whether nodes with the same color are more likely to be connected.

```
--verify_balanced_graph
```

Check if the graph is balanced based on the assigned edge signs. A balanced graph is one where the signs on the edges are consistent with the node attributes.

```
--output out_graph_file.gml
```

Specify the file to which the final graph and results should be saved. In this case, out_graph_file.gml is the output file that will receive the updated graph data—the node attributes and edge attributes should be also saved in the out_graph_file.gml.

Examples

-
  ```
  python ./graph_analysis.py graph_file.gml --components 3
  --plot C --output out_graph_file.gml
  ```

Read graph_file.gml and partition it into 3 connected components, plot the graph and highlight the clustering coefficient, and save the graph in out_graph_file.gml

- `python ./graph_analysis.py graph_file.gml --plot N --verify_homophily`

Read graph_file.gml, plot the graph and verify if there is evidence of homophily in the graph

Evaluation:

Your implementation will be evaluated based on correctness, efficiency, clarity of code, and the quality of graph visualization. Ensure your program handles edge cases such as empty graphs, non-existent files, and invalid inputs gracefully.

Submission:

Submit your source code files along with a README.md file detailing instructions on how to use your program.