**Project Documentation: Mini Social Networks Analysis Tool**

**Cover Page**

**Project Title:** Mini Social Networks Analysis Tool  
**Course Name:** Social Network Analysis  
**Team Members:**

1. Member 1 (ID: XXX) – Role: Project Manager & Developer
2. Member 2 (ID: XXX) – Role: Lead Developer
3. Member 3 (ID: XXX) – Role: UI/UX Designer
4. Member 4 (ID: XXX) – Role: Data Analyst
5. Member 5 (ID: XXX) – Role: Documentation & Testing
6. Member 6 (ID: XXX) – Role: Algorithm Specialist

**Submission Date:** [Date]

**Table of Contents**

1. **Introduction**
2. **System Requirements**
3. **Implementation Details**
   * Node and Edge Attributes
   * Basic Visualization
   * Layout Algorithms
   * Graph Metrics and Statistics
   * Filtering Options
   * Graph Partitioning and Clustering
   * Community Detection Comparison
   * Clustering Evaluation
   * Link Analysis Techniques
4. **Screenshots & Illustrations**
5. **Conclusion**

**1. Introduction**

This project is a **Mini Social Networks Analysis Tool** built using Python (Tkinter, NetworkX, PyVis). It allows users to:

* Load network data (nodes and edges from CSV).
* Visualize networks with customizable attributes.
* Apply different layout algorithms.
* Compute graph metrics.
* Filter nodes based on centrality and communities.
* Detect and compare communities.
* Perform link analysis (PageRank, Betweenness Centrality).

**2. System Requirements**

* **Python 3.8+**
* Libraries: networkx, pandas, pyvis, python-louvain, scikit-learn, tkinter
* Input: Two CSV files (nodes and edges).

**3. Implementation Details**

**3.1 Node and Edge Attributes**

Users can customize:

* **Node Size, Color, Label, Shape**
* **Edge Style (Solid, Dashed, Dotted)**

**Code Implementation:**

python

Copy

Download

def visualize\_network(self):

# Node styling

node\_attrs = {

"size": float(value) \* 2 if str(value).replace('.', '').isdigit() else 10,

"color": self.value\_to\_color(value),

"shape": selected\_shape,

"label": node\_label

}

# Edge styling

edge\_attrs = {

"width": float(value) / 2 if value else 1,

"dashes": True if selected\_style == "dashed" else [2, 5] if selected\_style == "dotted" else False

}

**Screenshot:**  
*(Insert screenshot of node/edge customization UI)*

**3.2 Basic Visualization**

* Displays nodes and edges.
* Supports directed/undirected graphs.

**Code Implementation:**

python

Copy

Download

net = Network(notebook=True, directed=(self.current\_graph\_type == "directed"))

net.from\_nx(g) # Convert NetworkX graph to PyVis

**Screenshot:**  
*(Insert screenshot of basic network visualization)*

**3.3 Layout Algorithms**

Implemented:

1. **Force-Directed (Fruchterman-Reingold)**
2. **Tree (Hierarchical)**
3. **Radial Layout**

**Code Implementation:**

python

Copy

Download

def generate\_layout(self, g, layout\_name, spring\_constant):

if layout\_name == "force-directed":

return nx.spring\_layout(g, k=spring\_constant / 100)

elif layout\_name == "tree":

return self.\_generate\_tree\_layout(g)

elif layout\_name == "radial":

return self.\_generate\_radial\_layout(g)

**Screenshot:**  
*(Insert comparison of different layouts)*

**3.4 Graph Metrics and Statistics**

Calculates:

* Degree distribution
* Clustering coefficient
* Average path length
* Assortativity

**Code Implementation:**

python

Copy

Download

def calculate\_basic\_metrics(self):

self.metrics\_text.insert(END, f"Degree Distribution: {degree\_count}\n")

self.metrics\_text.insert(END, f"Average Clustering: {nx.average\_clustering(g):.4f}\n")

self.metrics\_text.insert(END, f"Average Path Length: {nx.average\_shortest\_path\_length(g):.4f}\n")

**Screenshot:**  
*(Insert screenshot of metrics output)*

**3.5 Filtering Options**

**I. Centrality-Based Filtering**

* Degree, Betweenness, Eigenvector Centrality.

**Code Implementation:**

python

Copy

Download

def apply\_centrality\_filter(self):

if centrality\_type == "degree":

centrality = nx.degree\_centrality(g)

elif centrality\_type == "betweenness":

centrality = nx.betweenness\_centrality(g)

elif centrality\_type == "eigenvector":

centrality = nx.eigenvector\_centrality(g)

**Screenshot:**  
*(Insert screenshot of centrality filtering)*

**II. Community-Based Filtering**

Filters nodes by community membership.

**Code Implementation:**

python

Copy

Download

def apply\_community\_filter(self):

filtered\_nodes = [n for n in g.nodes() if self.node\_attributes[n]['community'] == selected\_community]

**Screenshot:**  
*(Insert screenshot of community filtering)*

**3.6 Graph Partitioning and Clustering**

Algorithms:

1. **Louvain (Modularity Optimization)**
2. **Girvan-Newman (Edge Betweenness)**

**Code Implementation:**

python

Copy

Download

def detect\_communities(self):

# Louvain

louvain\_partition = community\_louvain.best\_partition(g)

# Girvan-Newman

girvan\_communities = tuple(sorted(c) for c in next(girvan\_newman(g)))

**Screenshot:**  
*(Insert screenshot of community detection results)*

**3.7 Community Detection Comparison**

Compares:

* Number of communities
* Modularity
* Conductance

**Code Implementation:**

python

Copy

Download

nmi\_score = normalized\_mutual\_info\_score(louvain\_labels, girvan\_labels)

self.metrics\_text.insert(END, f"NMI Score: {nmi\_score:.4f}\n")

**Screenshot:**  
*(Insert comparison table)*

**3.8 Clustering Evaluation**

Metrics:

1. **Modularity**
2. **Conductance**
3. **Normalized Mutual Information (NMI)**

**Code Implementation:**

python

Copy

Download

girvan\_modularity = community\_louvain.modularity(girvan\_partition, g)

conductance = nx.algorithms.cuts.conductance(g, community\_nodes)

**Screenshot:**  
*(Insert evaluation metrics output)*

**3.9 Link Analysis Techniques**

* **PageRank**
* **Betweenness Centrality**

**Code Implementation:**

python

Copy

Download

def run\_pagerank(self):

pr = nx.pagerank(g)

def run\_betweenness(self):

betweenness = nx.betweenness\_centrality(g)

**Screenshot:**  
*(Insert PageRank & Betweenness results)*

**4. Screenshots & Illustrations**

*(Include annotated screenshots for each functionality)*

**4.1 Data Loading**

* Nodes CSV
* Edges CSV

**4.2 Visualization Customization**

* Node size/color
* Edge style

**4.3 Layout Algorithms**

* Force-directed
* Tree
* Radial

**4.4 Filtering Examples**

* Centrality-based
* Community-based

**4.5 Community Detection**

* Louvain vs. Girvan-Newman

**4.6 Link Analysis**

* PageRank output
* Betweenness output

**5. Conclusion**

This tool provides a **comprehensive** social network analysis solution with:

* **Customizable visualization**
* **Multiple layout options**
* **Advanced filtering**
* **Community detection & comparison**
* **Link analysis (PageRank, Betweenness)**

**Future Work:**

* Support for larger datasets
* Additional clustering algorithms (e.g., Label Propagation)

**End of Document**