In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

In [ ]:
```
#run this code to install the libraries, if you don't have them
!pip install networkx pandas plotly numpy matplotlib seaborn scikit-learn
```

In [2]:
```
#importing the necessary requirements
import networkx as nx
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import gzip
import shutil
import plotly.graph_objects as go
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
```

In [3]:
```
# Path to your downloaded file
file_path = '/content/drive/MyDrive/com-amazon.ungraph (1).txt.gz'

# Extract the compressed file
with gzip.open(file_path, 'rb') as f_in:
    with open('uncompressed_file.txt', 'wb') as f_out:
        shutil.copyfileobj(f_in, f_out)
```

In [4]:
```
#reading the ungrapgh dataset into a pandas dataframe
#and skipping the metadata lines and assigning column names
df = pd.read_csv("uncompressed_file.txt",
                 delimiter='\t',
                 skiprows=4,
                 names=['FromNodeId', 'ToNodeId'])
```

BASIC DATA EXPLORATION

In [ ]:
```
df.shape
```

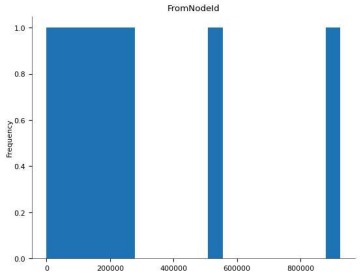
Out[ ]:
```
(925872, 2)
```

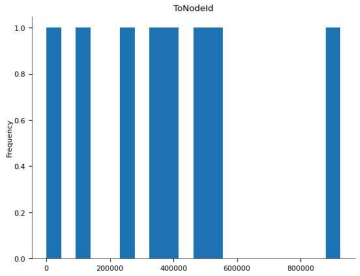
In [ ]:
```
df.describe()
```

Out[ ]:

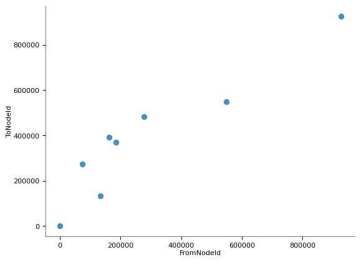
	FromNodeId	ToNodeId
count	925872.000000	925872.000000
mean	185662.827571	368949.221348
std	133061.964633	132601.362414
min	1.000000	366.000000
25%	73349.000000	273732.000000
50%	162058.000000	392319.000000
75%	277243.000000	482703.000000
max	548411.000000	548551.000000

Distributions

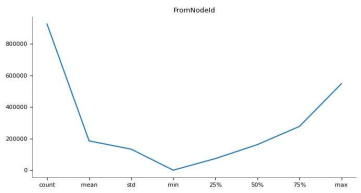


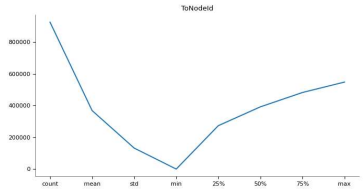


2-d distributions



Values





In [ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 925872 entries, 0 to 925871
Data columns (total 2 columns):
 #   Column      Non-Null Count   Dtype
---  ------      --------------   -----
 0   FromNodeId  925872 non-null  int64
 1   ToNodeId    925872 non-null  int64
dtypes: int64(2)
memory usage: 14.1 MB
```

In [ ]:
```
df.head()
```

Out[ ]:

	FromNodeId	ToNodeId
0	1	88160
1	1	118052
2	1	161555
3	1	244916
4	1	346495

In [ ]:
```
df.FromNodeId.nunique()
```

Out[ ]: 265933

In [ ]:
```
df.ToNodeId.nunique()
```

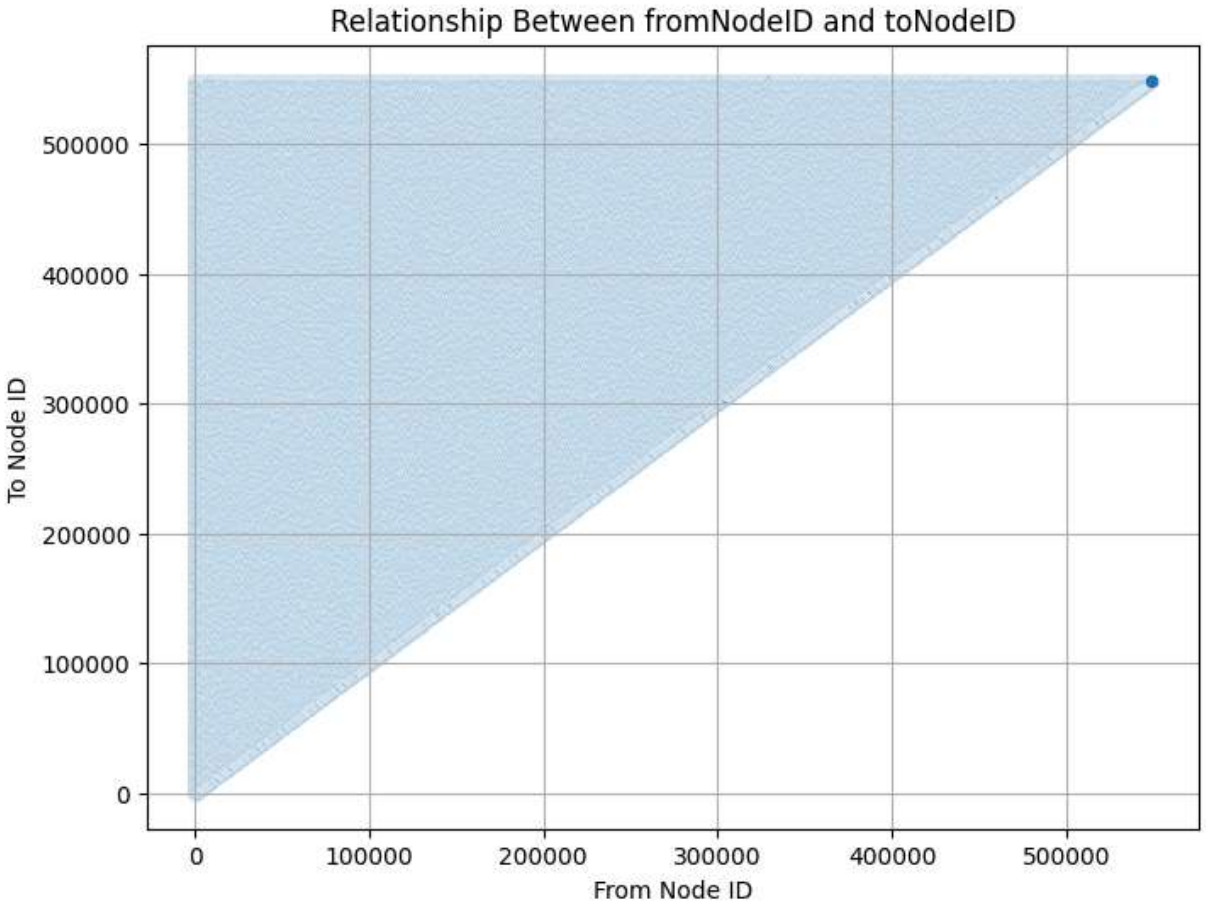
Out[ ]: 264147

In [ ]:
```
duplicates = df[df.duplicated()]
print(duplicates)
```

```
Empty DataFrame
Columns: [FromNodeId, ToNodeId]
Index: []
```

In [ ]:
```
# Create figure
plt.figure(figsize=(8, 6))
sns.scatterplot(x=df['FromNodeId'], y=df['ToNodeId'], alpha=0.7)

plt.xlabel("From Node ID")
plt.ylabel("To Node ID")
plt.title("Relationship Between fromNodeID and toNodeID")
plt.grid(True)
```

```
# Get current figure and save it
fig = plt.gcf()  # Get current figure
fig.savefig("node_relationship.png")  # Save the figure

# Show the plot
plt.show()
```



RESEARCH QUESTION ONE:What are the largest connected components, and how fragmented is the network?

In [ ]:
```
# Create an undirected graph
G = nx.from_pandas_edgelist(df, 'FromNodeId', 'ToNodeId')

# Get connected components
components = sorted(nx.connected_components(G), key=len, reverse=True)

# Largest connected component
largest_cc = G.subgraph(components[0])

#checking how fragmented the network is.
num_components = nx.number_connected_components(G)
print(f"Number of connected components: {num_components}")

# Size of largest connected component
largest_cc = len(components[0])
print(f"Largest component size: {largest_cc}")
```

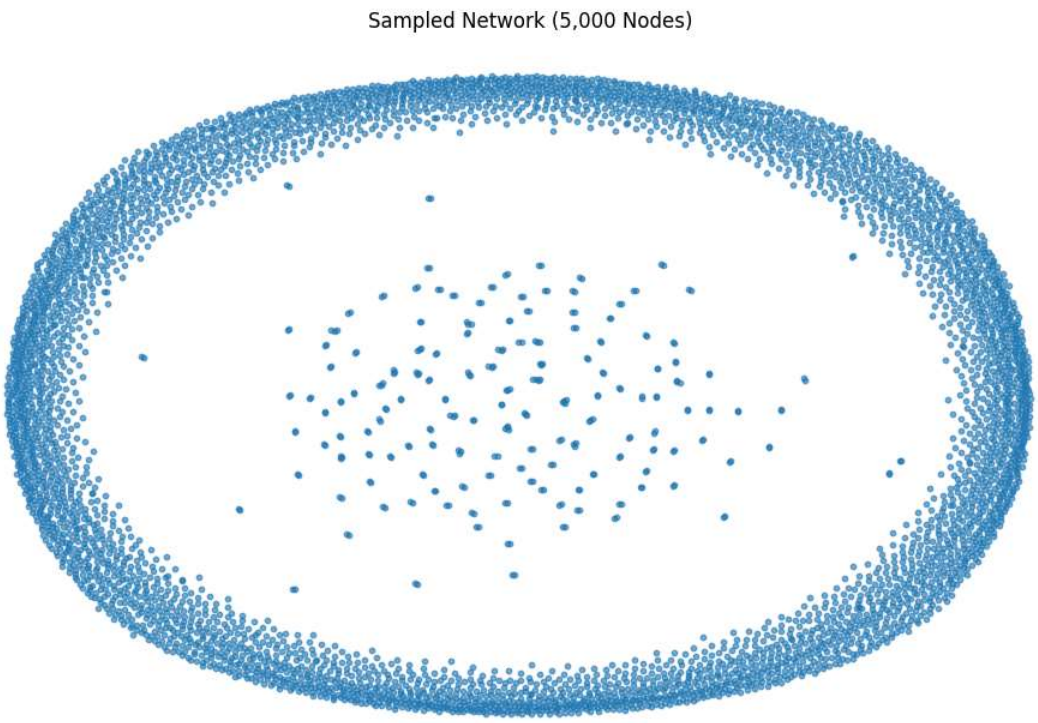
```
#fragmentation index
fragmentation = 1 - (largest_cc / G.number_of_nodes())
print(f"Network fragmentation index: {fragmentation:.2f}")

#checking to confirm if the network is fully connected
if nx.is_connected(G):
    print("The network is fully connected (1 component).")
else:
    print(f"The network has {len(components)} connected components.")


#computing the average component size
avg_component_size = sum(len(c) for c in components) / len(components)
print(f"Average component size: {avg_component_size:.2f}")

#checking the network to determine the plot
print(f"Nodes: {G.number_of_nodes()}")
print(f"Edges: {G.number_of_edges()}")

#Computing the Network Density
density = nx.density(G)
print(f"Network Density: {density:.6f}")

#computing Average Clustering Coefficient
clustering_coeff = nx.average_clustering(G)
print(f"Average Clustering Coefficient: {clustering_coeff:.4f}")
```

```
Number of connected components: 1
Largest component size: 334863
Network fragmentation index: 0.00
The network is fully connected (1 component).
Average component size: 334863.00
Nodes: 334863
Edges: 925872
Network Density: 0.000017
Average Clustering Coefficient: 0.3967
```

In [ ]:

```
import random
#Sample 5000 random nodes
sample_nodes = random.sample(list(G.nodes), min(5000, len(G.nodes)))
G_sample = G.subgraph(sample_nodes)

# Plot sampled graph
plt.figure(figsize=(10, 6))
pos = nx.spring_layout(G_sample, seed=42)
nx.draw(G_sample, pos, node_size=10, edge_color="gray", alpha=0.6)

# Title
plt.title("Sampled Network (5,000 Nodes)")


# Save the figure as a high-quality image
plt.savefig("Sampled Network (5,000 Nodes).png")
```

```
#show the plot
plt.show()
```



RESEARCH QUESTION 2: How does degree centrality score vary across different node?

In [ ]:
```
# Create an undirected graph
G = nx.from_pandas_edgelist(df, 'FromNodeId', 'ToNodeId')

# Compute degree centrality
degree_centrality = nx.degree_centrality(G)

# Convert to DataFrame
degree_df = pd.DataFrame(degree_centrality.items(), columns=['NodeId', 'Degree_Cent
```

In [ ]:
```
degree_df.shape
```

Out[ ]: (334863, 2)

In [ ]:
```
degree_df.describe()
```

Out[ ]:

	NodeId	Degree_Centrality
count	334863.000000	334863.000000
mean	276768.565727	0.000017
std	159927.553896	0.000017
min	1.000000	0.000003
25%	138028.000000	0.000009
50%	276405.000000	0.000012
75%	415626.500000	0.000018
max	548551.000000	0.001639

In [ ]:
```
degree_df.head()
```

Out[ ]:

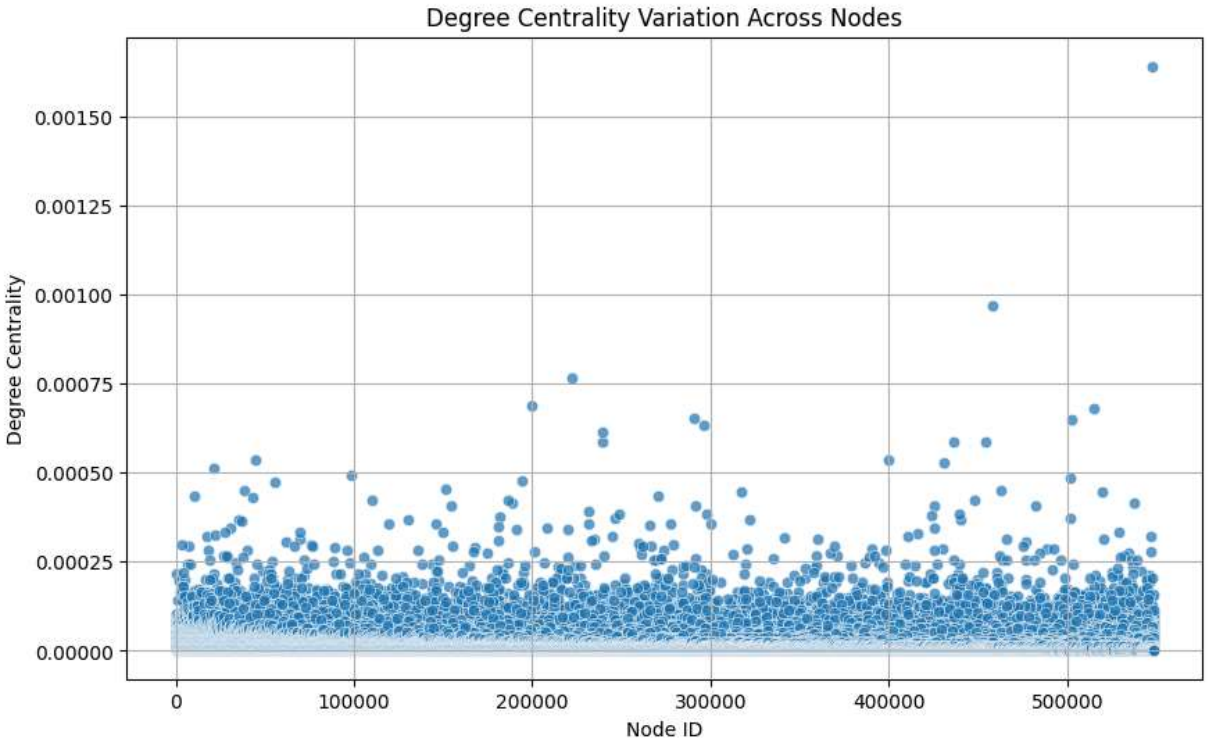
	NodeId	Degree_Centrality
0	1	0.000024
1	88160	0.000021
2	118052	0.000054
3	161555	0.000093
4	244916	0.000078

In [ ]:
```
degree_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 334863 entries, 0 to 334862
Data columns (total 2 columns):
 #   Column             Non-Null Count   Dtype  
---  ------             --------------   -----  
 0   NodeId             334863 non-null  int64  
 1   Degree_Centrality  334863 non-null  float64
dtypes: float64(1), int64(1)
memory usage: 5.1 MB
```

In [ ]:
```
plt.figure(figsize=(10, 6))

# Scatter plot
sns.scatterplot(x=degree_df['NodeId'], y=degree_df['Degree_Centrality'], alpha=0.7)

# Labels and title
plt.xlabel("Node ID")
plt.ylabel("Degree Centrality")
plt.title("Degree Centrality Variation Across Nodes")
plt.grid(True)

# Save the figure
```

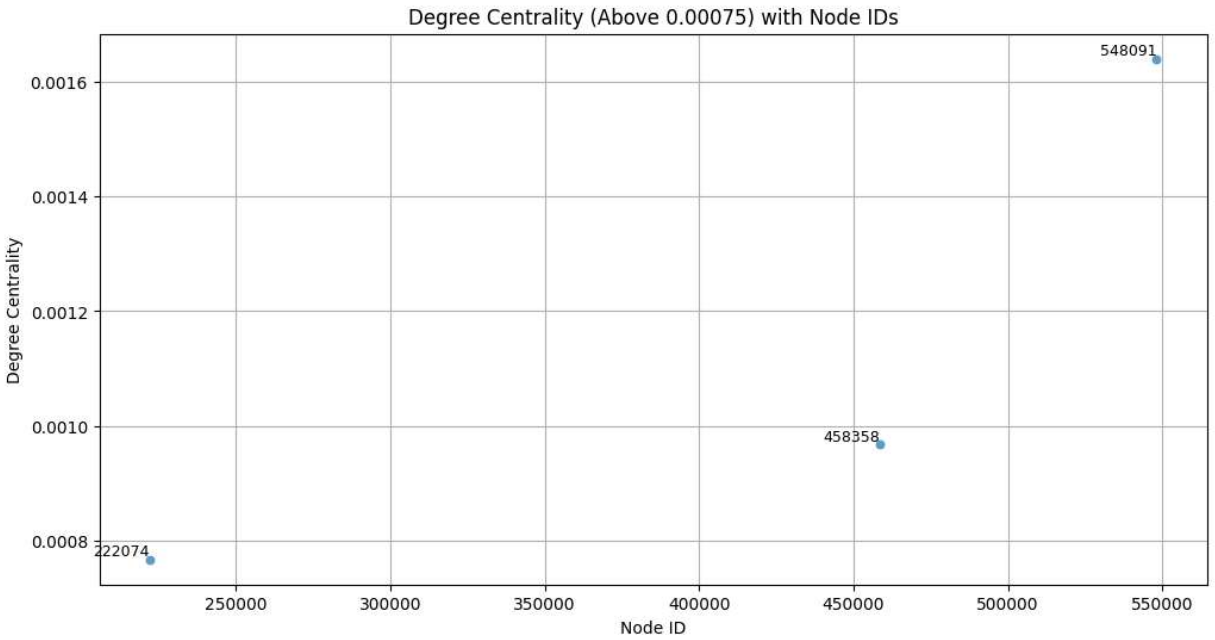
```
plt.savefig("degree_centrality_variation.png", dpi=300, bbox_inches="tight")  # Sav

# Show the plot
plt.show()
```



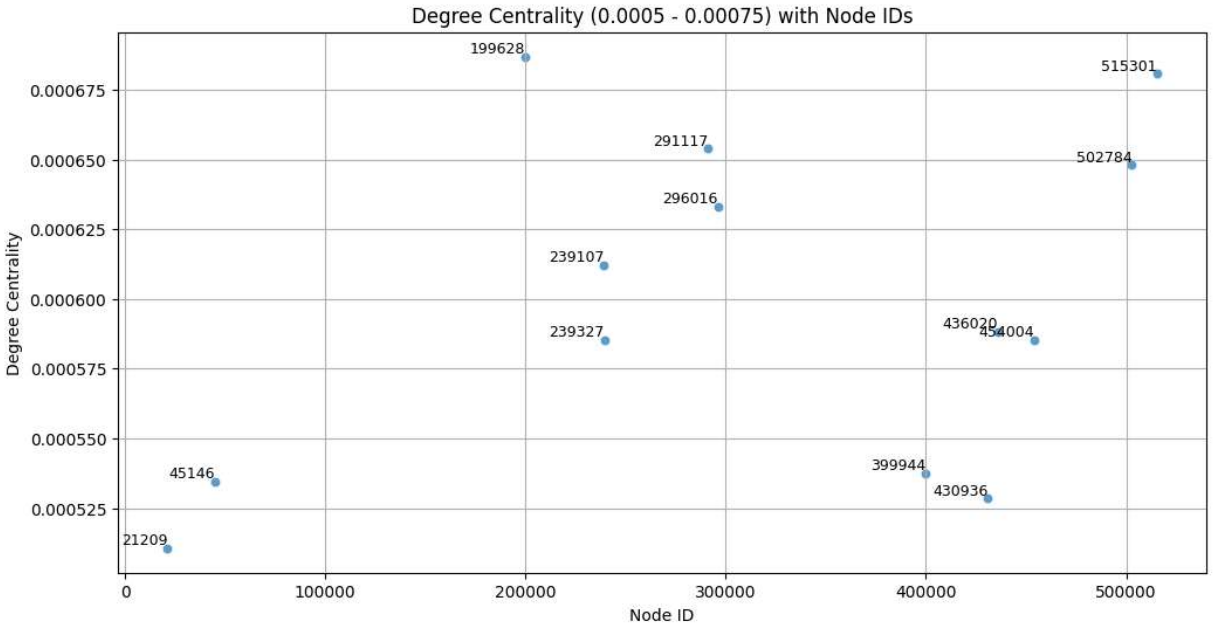
In [ ]:

```
import matplotlib.pyplot as plt
import seaborn as sns

# Filter the dataset to include only Degree_Centrality > 0.00075
filtered_df = degree_df[degree_df['Degree_Centrality'] > 0.00075]

plt.figure(figsize=(12, 6))

# Scatter plot with filtered data
sns.scatterplot(x=filtered_df['NodeId'], y=filtered_df['Degree_Centrality'], alpha=

# Annotate each point with its Node ID
for i in range(len(filtered_df)):
    plt.text(filtered_df['NodeId'].iloc[i],
             filtered_df['Degree_Centrality'].iloc[i],
             str(filtered_df['NodeId'].iloc[i]),
             fontsize=9, ha='right', va='bottom')

# Labels and title
plt.xlabel("Node ID")
plt.ylabel("Degree Centrality")
plt.title("Degree Centrality (Above 0.00075) with Node IDs")
plt.grid(True)

# Save the figure
plt.savefig("degree_centrality_with_labels.png")
```

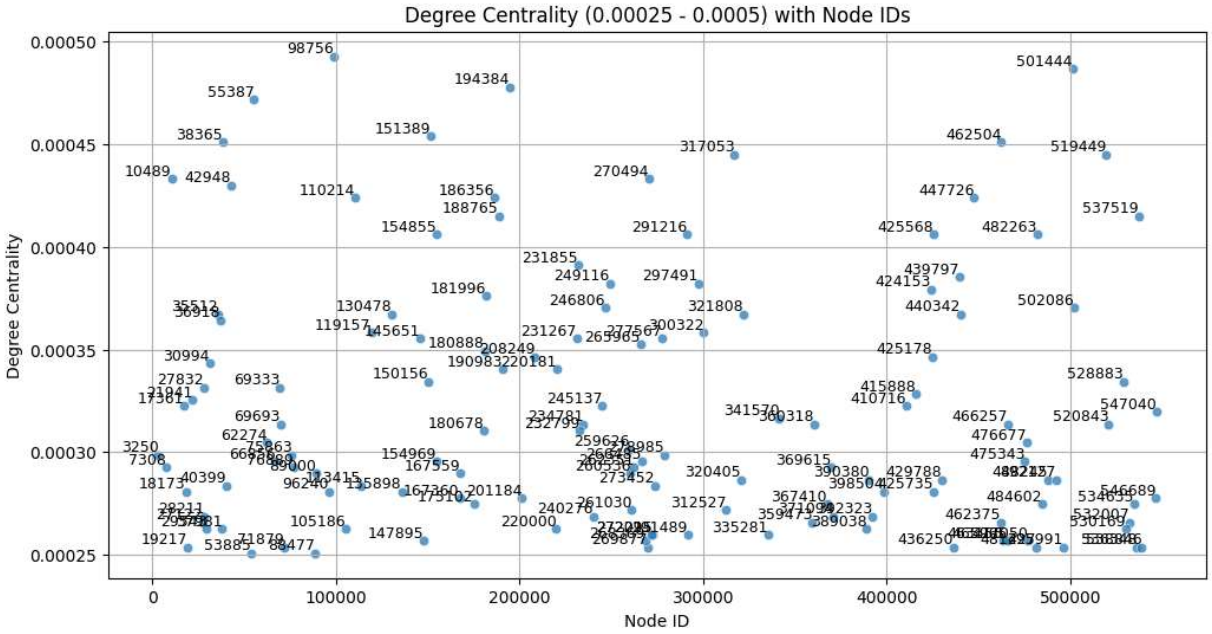
```
# Show the plot
plt.show()
```



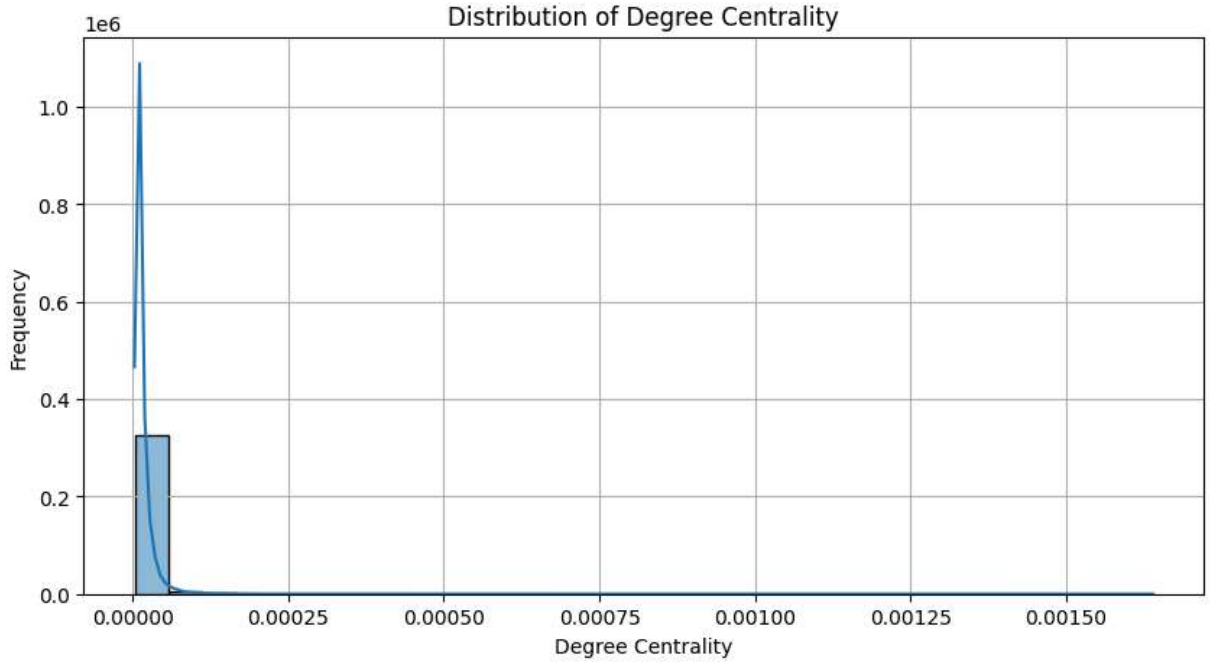
In [ ]:
```
# Filter nodes with Degree Centrality between 0.0005 and 0.00075
filtered_df = degree_df[(degree_df['Degree_Centrality'] > 0.0005) &
                        (degree_df['Degree_Centrality'] <= 0.00075)]

plt.figure(figsize=(12, 6))

# Scatter plot with filtered data
sns.scatterplot(x=filtered_df['NodeId'], y=filtered_df['Degree_Centrality'], alpha=

# Annotate each point with its Node ID
for i in range(len(filtered_df)):
    plt.text(filtered_df['NodeId'].iloc[i],
             filtered_df['Degree_Centrality'].iloc[i],
             str(filtered_df['NodeId'].iloc[i]),
             fontsize=9, ha='right', va='bottom')

# Labels and title
plt.xlabel("Node ID")
plt.ylabel("Degree Centrality")
plt.title("Degree Centrality (0.0005 - 0.00075) with Node IDs")
plt.grid(True)

# Save the figure
plt.savefig("degree_centrality_range_0005_00075.png", dpi=300, bbox_inches="tight")

# Show the plot
plt.show()
```



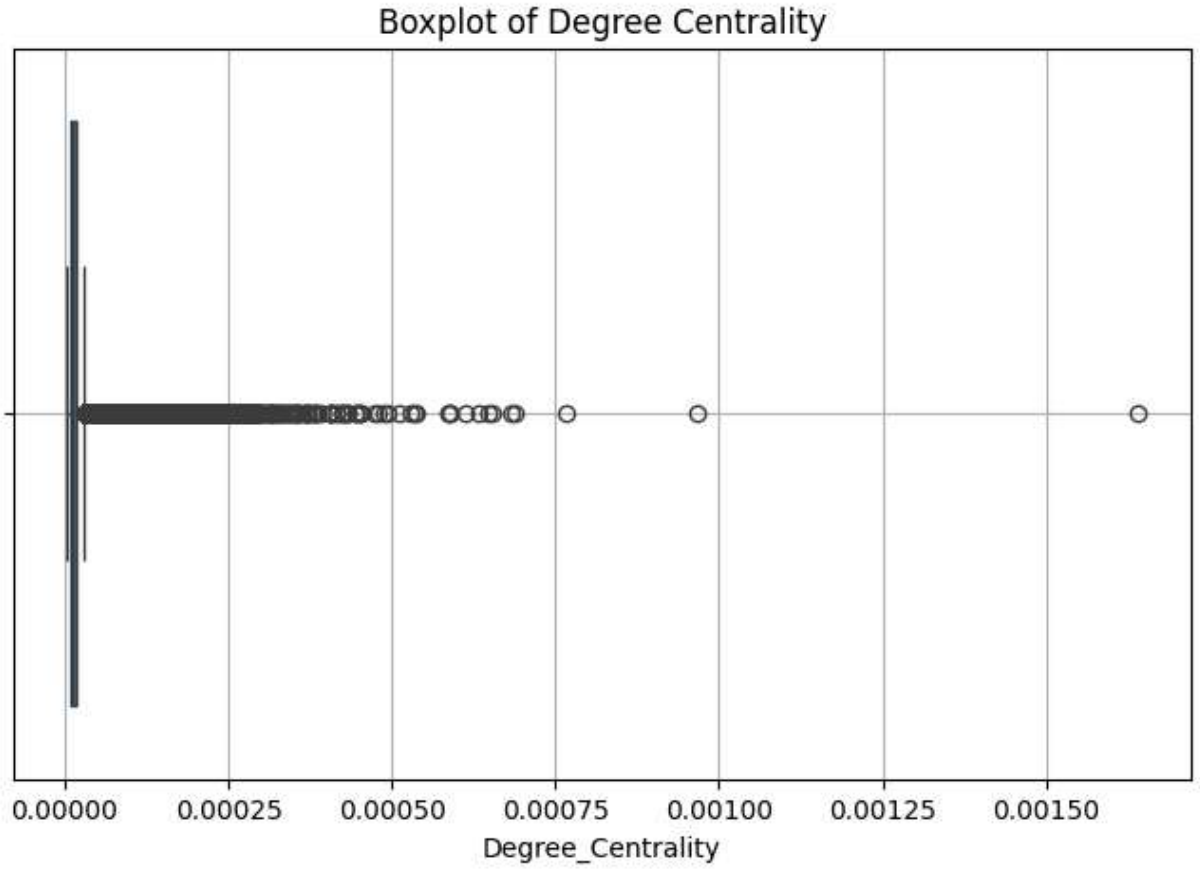
In [ ]:
```
# Filter nodes with Degree Centrality between 0.00025 and 0.0005
filtered_df = degree_df[(degree_df['Degree_Centrality'] >= 0.00025) &
                        (degree_df['Degree_Centrality'] < 0.0005)]

plt.figure(figsize=(12, 6))

# Scatter plot with filtered data
sns.scatterplot(x=filtered_df['NodeId'], y=filtered_df['Degree_Centrality'], alpha=

# Annotate each point with its Node ID
for i in range(len(filtered_df)):
    plt.text(filtered_df['NodeId'].iloc[i],
             filtered_df['Degree_Centrality'].iloc[i],
             str(filtered_df['NodeId'].iloc[i]),
             fontsize=9, ha='right', va='bottom')

# Labels and title
plt.xlabel("Node ID")
plt.ylabel("Degree Centrality")
plt.title("Degree Centrality (0.00025 - 0.0005) with Node IDs")
plt.grid(True)

# Save the figure
plt.savefig("degree_centrality_range_0005_00075.png", dpi=300, bbox_inches="tight")

# Show the plot
plt.show()
```



In [ ]:
```
# Create a histogram with KDE to see the overall distribution of degree centrality
plt.figure(figsize=(10, 5))
sns.histplot(degree_df['Degree_Centrality'], bins=30, kde=True)
plt.xlabel("Degree Centrality")
plt.ylabel("Frequency")
plt.title("Distribution of Degree Centrality")
plt.grid(True)

# Save the histogram as a high-quality image
plt.savefig("degree_centrality_distribution.png", dpi=300, bbox_inches="tight")

# Show the plot
plt.show()
```

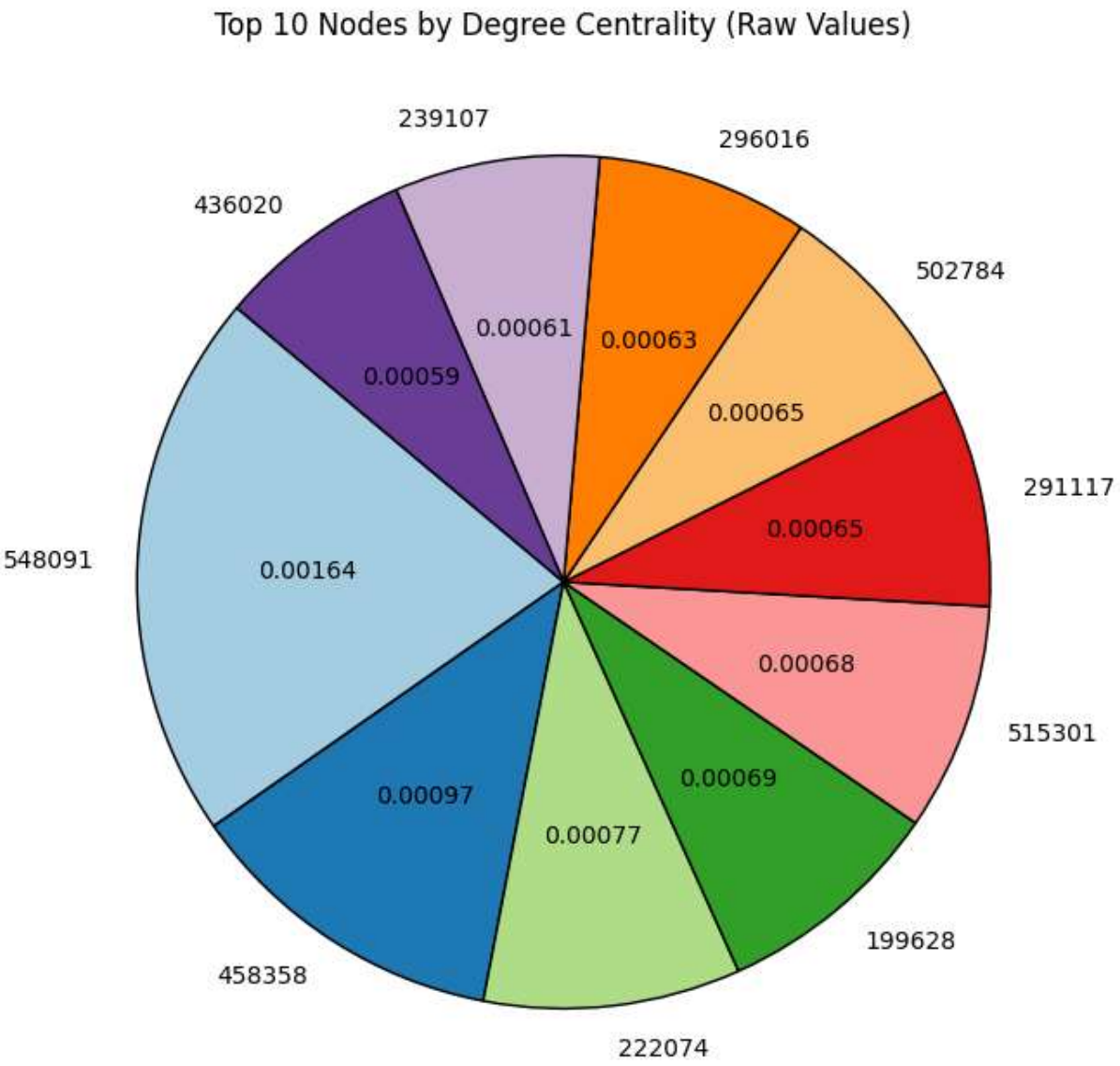


In [ ]:
```
#Using a boxplot to see outliers and general distribution.
plt.figure(figsize=(8, 5))
sns.boxplot(x=degree_df['Degree_Centrality'])
plt.title("Boxplot of Degree Centrality")
plt.grid(True)
plt.show()
```



In [ ]:
```
import matplotlib.pyplot as plt

# Select the top 10 nodes with the highest degree centrality
top_nodes = degree_df.nlargest(10, 'Degree_Centrality')

# Create labels using Node IDs
labels = top_nodes['NodeId'].astype(str)  # Convert Node IDs to string for labeling

# Define autopct function to display actual centrality values
def centrality_autopct(pct, all_values):
    """Function to format pie chart labels with actual degree centrality values."""
    absolute = pct / 100. * sum(all_values)
    return f"{absolute:.5f}"  # Format centrality value to 5 decimal places

# Create a pie chart
plt.figure(figsize=(8, 8))
plt.pie(top_nodes['Degree_Centrality'], labels=labels,
        autopct=lambda pct: centrality_autopct(pct, top_nodes['Degree_Centrality']),
        startangle=140, colors=plt.cm.Paired.colors, wedgeprops={'edgecolor': 'blac

# Title
```

```
plt.title("Top 10 Nodes by Degree Centrality (Raw Values)")

# Save the figure as a high-quality image
plt.savefig("top_10_nodes_degree_centrality_pie_raw_values.png", dpi=300, bbox_inch

# Show the plot
plt.show()
```



RESEARCH QUESTION 3: How does clustering coefficient distribution vary across nodes, and what does it reveal about local connectivity?

In [ ]:
```
# Create an undirected graph
G = nx.from_pandas_edgelist(df, 'FromNodeId', 'ToNodeId')

# Compute clustering coefficient for each node
clustering_coeffs = nx.clustering(G)
```

```
# Convert to DataFrame
clustering_df = pd.DataFrame({
    'NodeId': list(clustering_coeffs.keys()),
    'Clustering_Coefficient': list(clustering_coeffs.values())
})
```

In [ ]:
```
clustering_df.shape
```

Out[ ]: (334863, 2)

In [ ]:
```
clustering_df.describe()
```

Out[ ]:

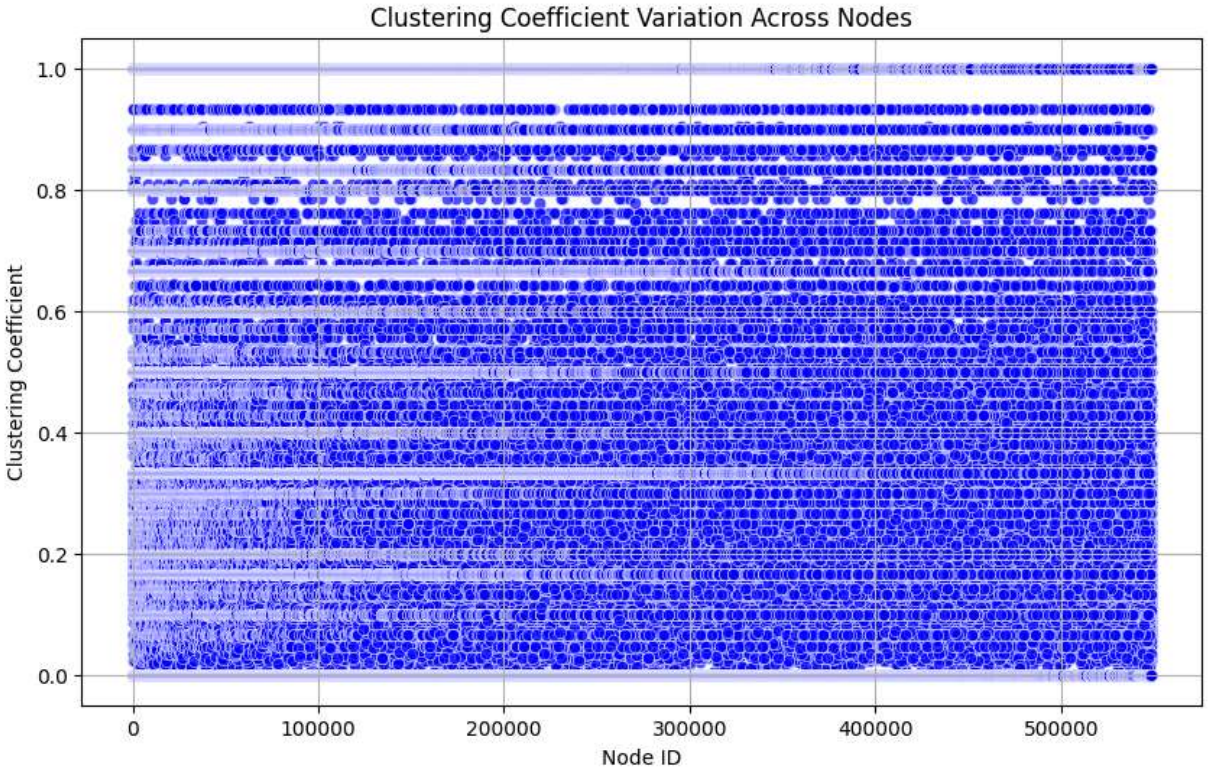
	NodeId	Clustering_Coefficient
count	334863.000000	334863.000000
mean	276768.565727	0.396746
std	159927.553896	0.329530
min	1.000000	0.000000
25%	138028.000000	0.100000
50%	276405.000000	0.333333
75%	415626.500000	0.666667
max	548551.000000	1.000000

In [ ]:
```
clustering_df.info()
```

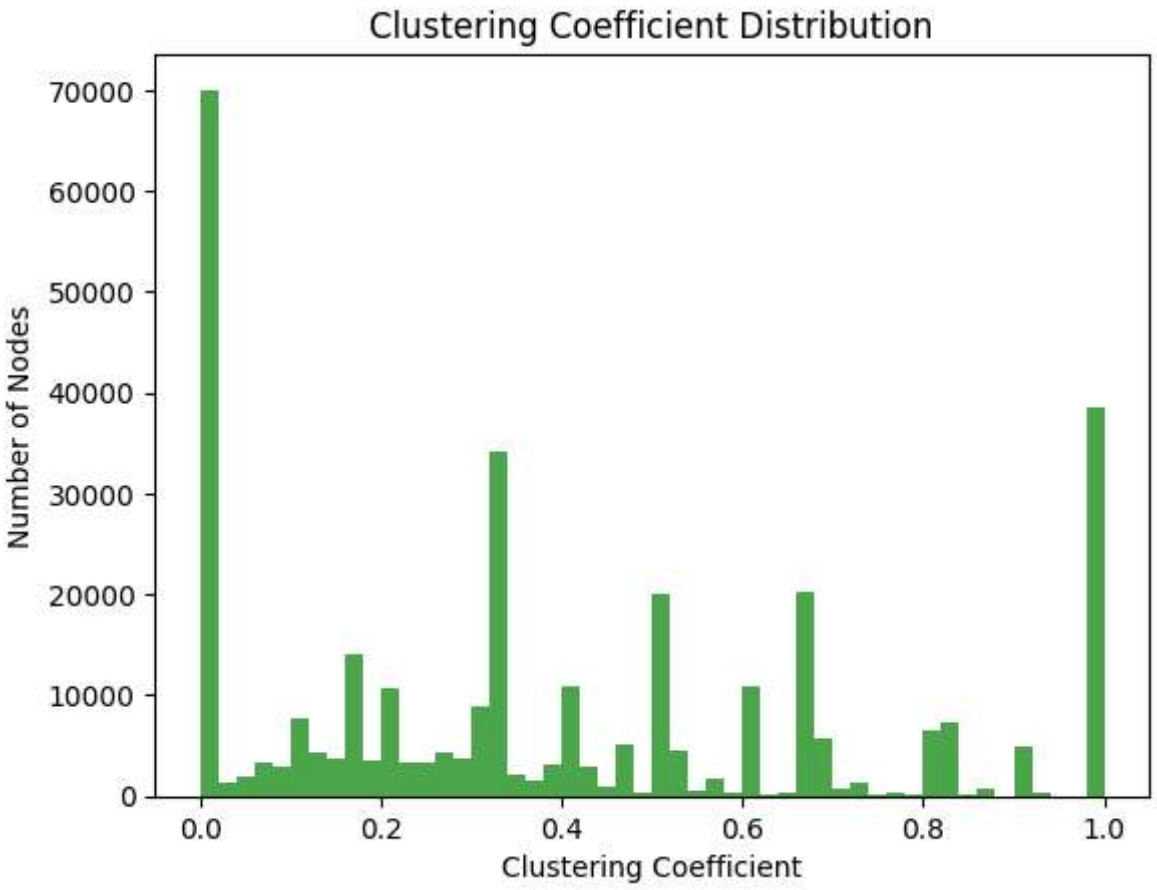
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 334863 entries, 0 to 334862
Data columns (total 2 columns):
 #   Column                 Non-Null Count   Dtype  
---  ------                 --------------   -----  
 0   NodeId                 334863 non-null  int64  
 1   Clustering_Coefficient  334863 non-null  float64
dtypes: float64(1), int64(1)
memory usage: 5.1 MB
```

In [ ]:
```
# Scatter plot
plt.figure(figsize=(10, 6))
sns.scatterplot(x=clustering_df['NodeId'], y=clustering_df['Clustering_Coefficient'

# Labels and title
plt.xlabel("Node ID")
plt.ylabel("Clustering Coefficient")
plt.title("Clustering Coefficient Variation Across Nodes")
plt.grid(True)

# Save the figure
plt.savefig("clustering_coefficient_variation.png", dpi=300, bbox_inches="tight")
```

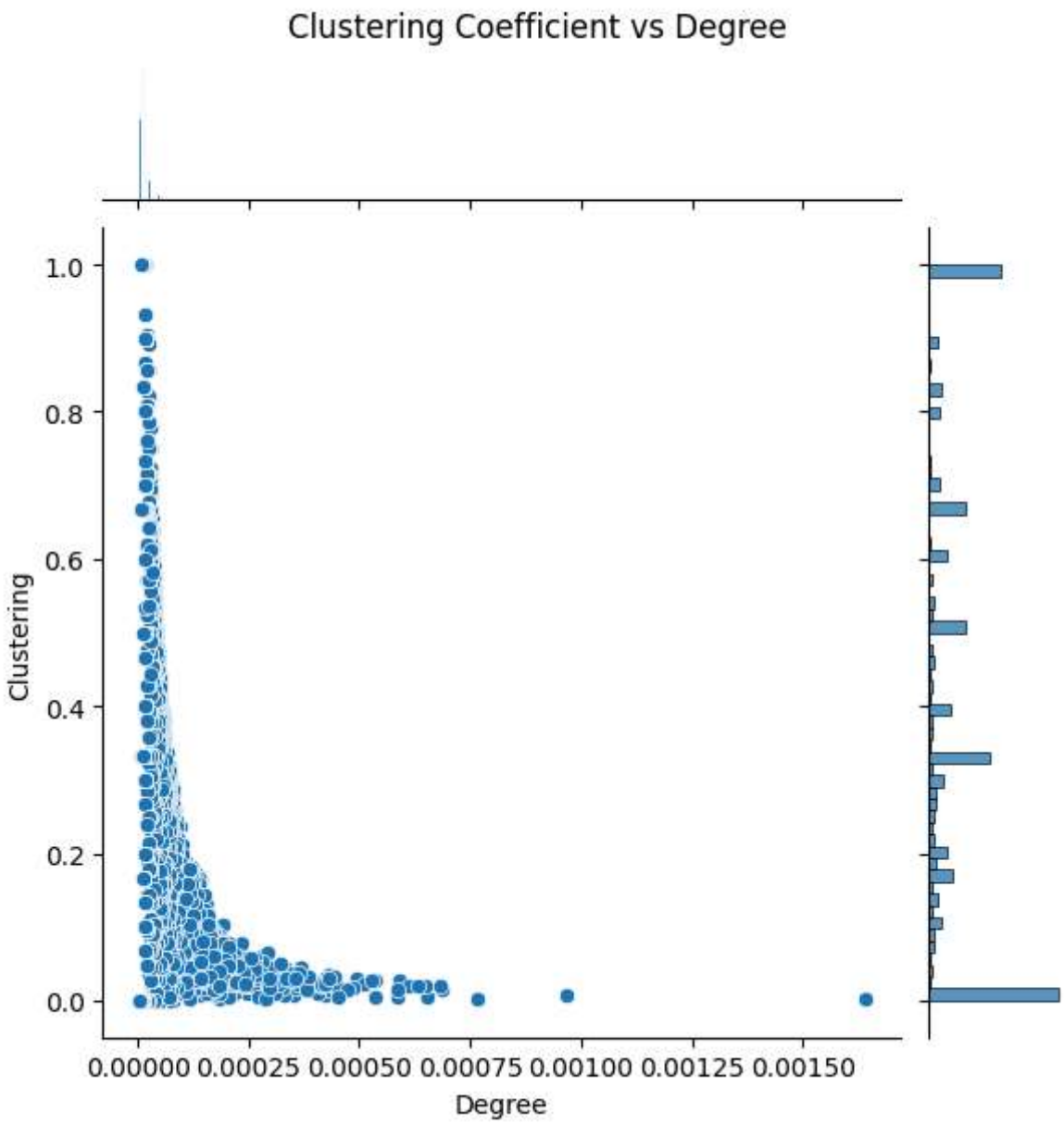
```
# Show the plot
plt.show()
```



In [ ]:
```
clustering_coeffs = list(nx.clustering(G).values())

plt.hist(clustering_coeffs, bins=50, color='green', alpha=0.7)
plt.xlabel("Clustering Coefficient")
plt.ylabel("Number of Nodes")
plt.title("Clustering Coefficient Distribution")

# Save the figure as a high-quality image
plt.savefig("Clustering Coefficient Distribution.png")

plt.show()
```

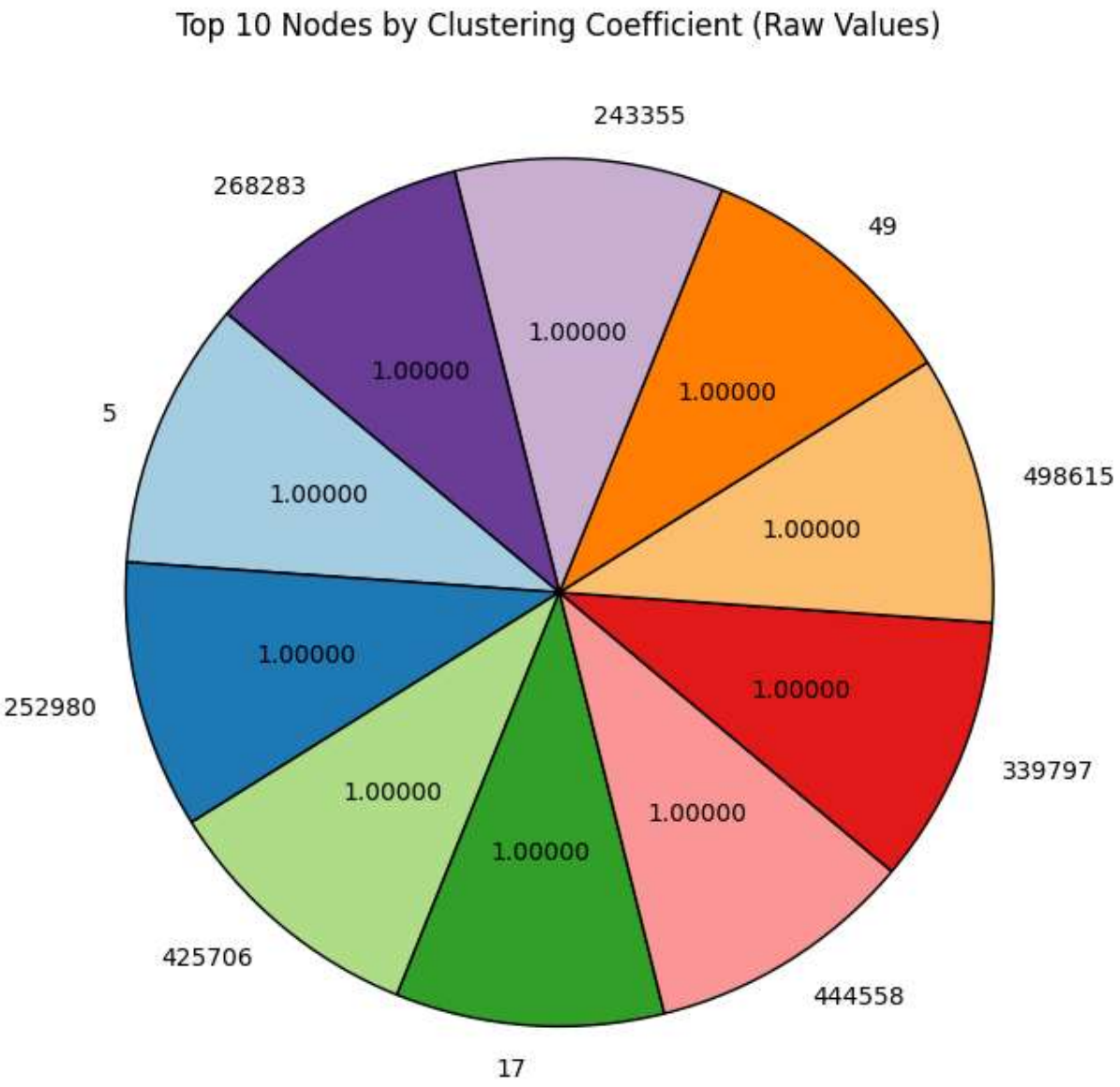


In [ ]:
```
#Create an undirected graph
G = nx.from_pandas_edgelist(df, 'FromNodeId', 'ToNodeId')

degree_centrality = nx.degree_centrality(G)
clustering = nx.clustering(G)

df_2 = pd.DataFrame({'Degree': degree_centrality, 'Clustering': clustering})

#creating a jointplot using seaborn
g = sns.jointplot(data=df_2, x='Degree', y='Clustering')

#adding a title
g.fig.suptitle('Clustering Coefficient vs Degree',  y=1.02)

#adding the figure to your system
g.fig.savefig('joint.png')
plt.show()
```



In [ ]:
```python
# Select the top 10 nodes with the highest clustering coefficient
top_nodes = clustering_df.nlargest(10, 'Clustering_Coefficient')

# Create labels using Node IDs
labels = top_nodes['NodeId'].astype(str)  # Convert Node IDs to string for labeling

# Define autopct function to display actual clustering coefficient values
def clustering_autopct(pct, all_values):
    """Function to format pie chart labels with actual clustering coefficient value
    absolute = pct / 100. * sum(all_values)
    return f"{absolute:.5f}"  # Format coefficient value to 5 decimal places

# Create a pie chart
plt.figure(figsize=(8, 8))
plt.pie(top_nodes['Clustering_Coefficient'], labels=labels,
        autopct=lambda pct: clustering_autopct(pct, top_nodes['Clustering_Coefficie
        startangle=140, colors=plt.cm.Paired.colors, wedgeprops={'edgecolor': 'blac

# Title
```

```
plt.title("Top 10 Nodes by Clustering Coefficient (Raw Values)")

# Save the figure as a high-quality image
plt.savefig("top_10_nodes_clustering_coefficient_pie_raw_values.png", dpi=300, bbox

# Show the plot
plt.show()
```



REASEARCH 4: What is the redundancy of connections, and how does it affect robustness?

removing 10% highest degree nodes

In [ ]:
```
# Identify top 10% highest-degree nodes
degree_centrality = nx.degree_centrality(G)
top_nodes = sorted(degree_centrality, key=degree_centrality.get, reverse=True)[:int

# Remove them from graph
```

```
G_removed = G.copy()
G_removed.remove_nodes_from(top_nodes)

# Check impact on connectivity
components_after = list(nx.connected_components(G_removed))
print(f"Remaining Components after Removal: {len(components_after)}")

# Check if network is still connected
is_still_connected = nx.is_connected(G_removed)
print(f"After removing 10% of Highest degree nodes, network still connected? {is_st

# Find largest remaining connected component
largest_cc_after_removal = max(nx.connected_components(G_removed), key=len)
print(f"Largest remaining component size: {len(largest_cc_after_removal)}")

#fragmentation index
fragmentation = 1 - (len(largest_cc_after_removal) / G_removed.number_of_nodes())
print(f"Network fragmentation index: {fragmentation:.2f}")

#computing the average component size
avg_component_size = sum(len(c) for c in components_after) / len(components_after)
print(f"Average component size: {avg_component_size:.2f}")

#checking the network to determine the plot
print(f"Nodes: {G_removed.number_of_nodes()}")
print(f"Edges: {G_removed.number_of_edges()}")

#Computing the Network Density
density = nx.density(G_removed)
print(f"Network Density: {density:.6f}")

#computing Average Clustering Coefficient
clustering_coeff = nx.average_clustering(G_removed)
print(f"Average Clustering Coefficient: {clustering_coeff:.4f}")
```

```
Remaining Components after Removal: 47453
After removing 10% of Highest degree nodes, network still connected? False
Largest remaining component size: 210170
Network fragmentation index: 0.30
Average component size: 6.35
Nodes: 301377
Edges: 409930
Network Density: 0.000009
Average Clustering Coefficient: 0.2660
```

In [ ]:
```
# Get all connected components and their sizes
components = sorted(nx.connected_components(G_removed), key=len, reverse=True)

# Create a DataFrame
df_components = pd.DataFrame({
    "Component": [f"Component {i+1}" for i in range(len(components))],
    "Size": [len(comp) for comp in components]
})
```

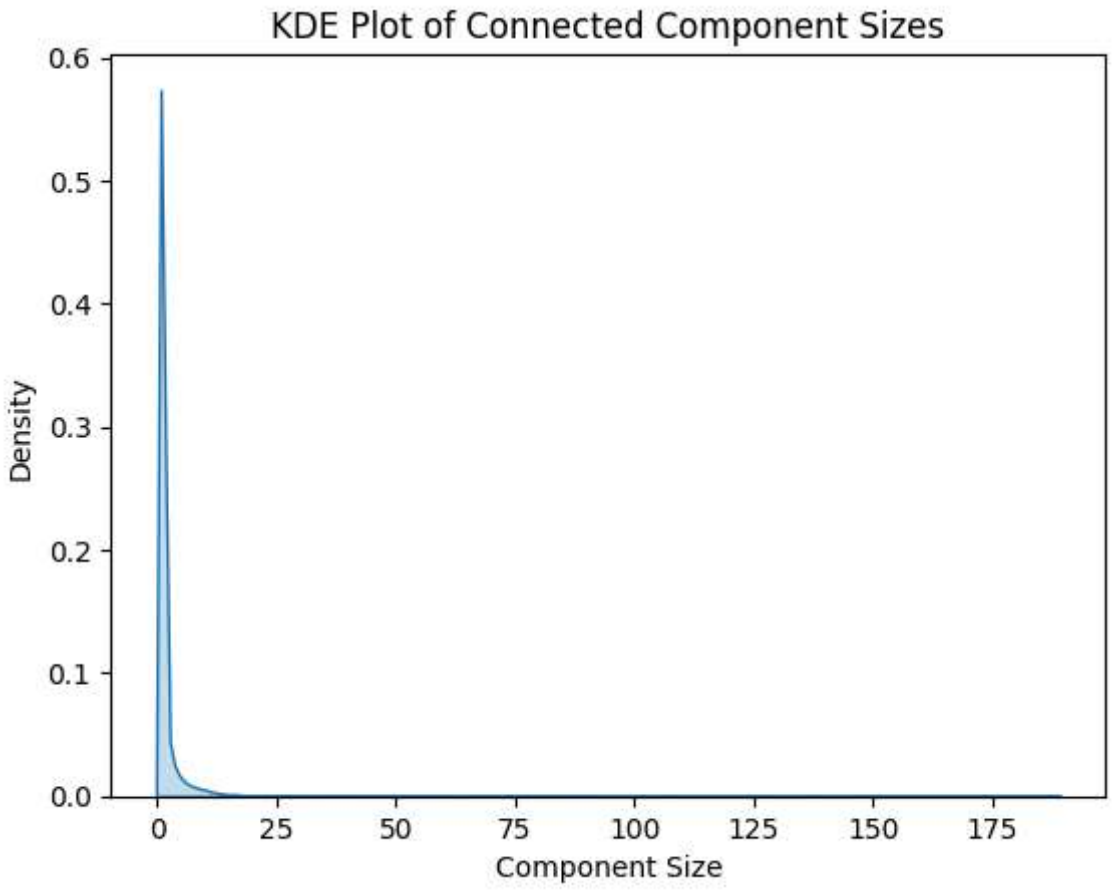
In [ ]:
```
df_components.describe()
```

Out[ ]:

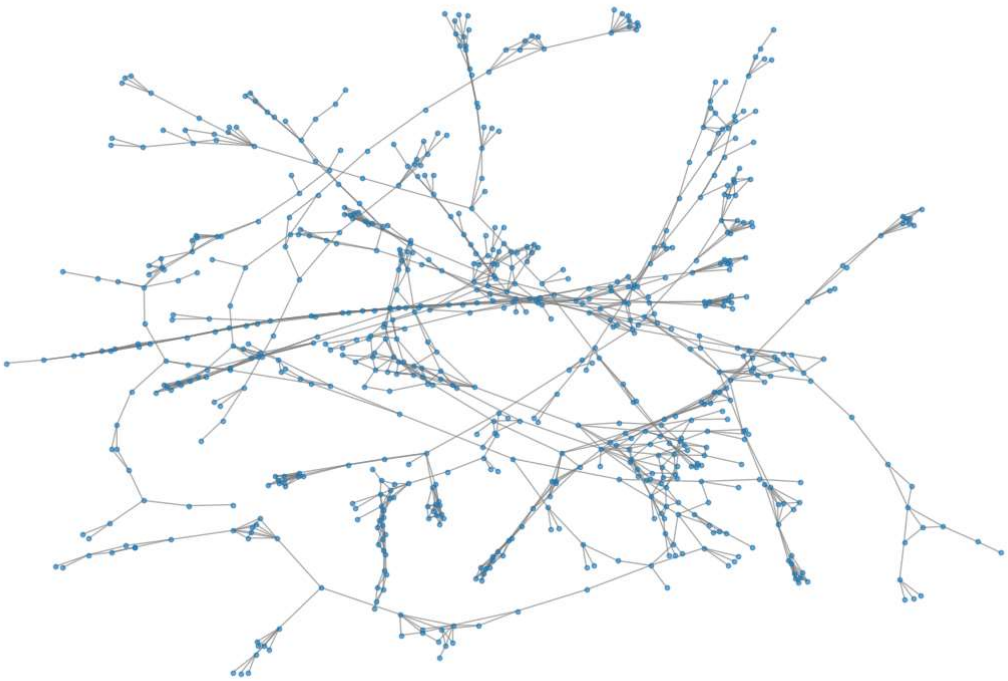
	Size
count	47453.000000
mean	6.351063
std	964.801583
min	1.000000
25%	1.000000
50%	1.000000
75%	1.000000
max	210170.000000

In [ ]:

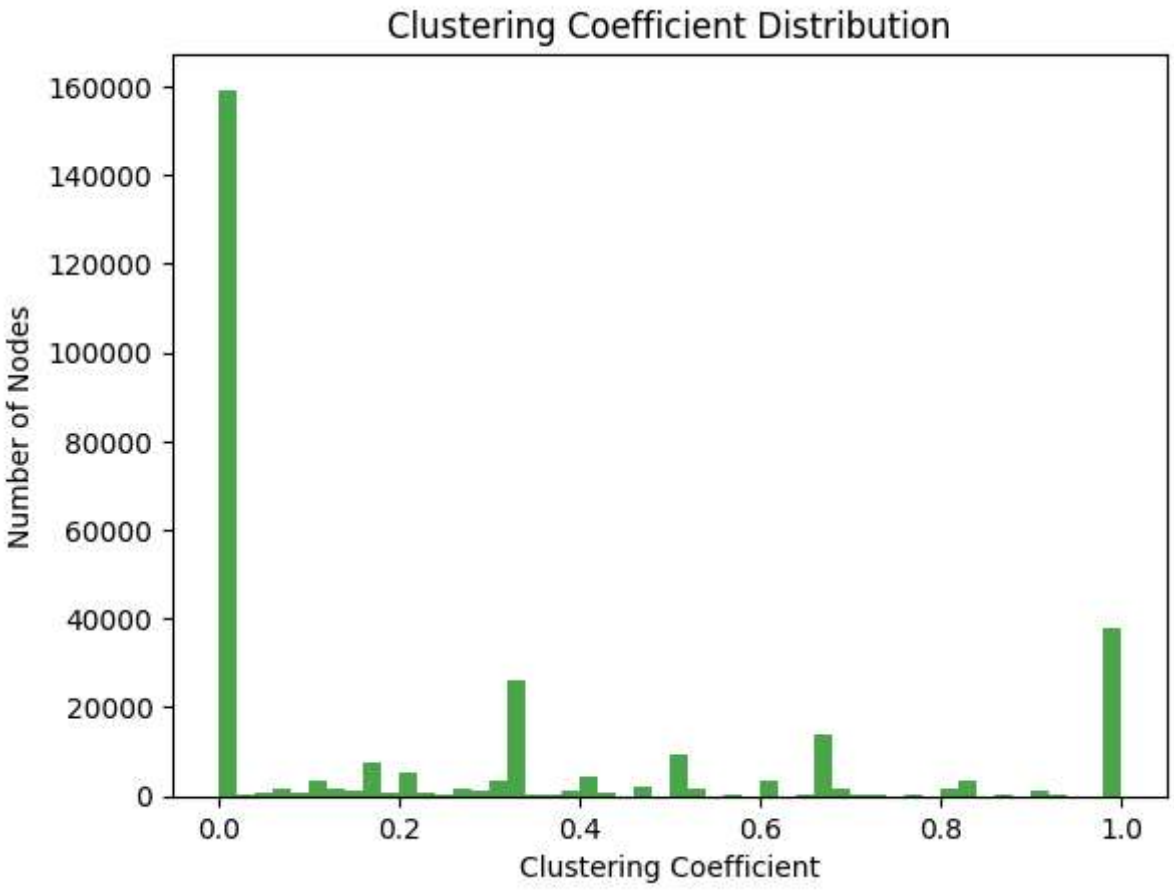
```
# Create the KDE plot with fill=True for a filled area under the curve
filtered_df = df_components[df_components['Size'] < 50000]  # Example: Filter out t
sns.kdeplot(filtered_df['Size'], fill=True)

# Customize the plot
plt.xlabel("Component Size")  # Set x-axis label
plt.ylabel("Density")  # Set y-axis label
plt.title("KDE Plot of Connected Component Sizes")  # Set plot title

# Display the plot
plt.show()
```



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

# Find all connected components with 100-200 nodes
medium_components = [comp for comp in nx.connected_components(G_removed) if 100 <=

if medium_components:
    # Merge all selected subgraphs into one graph
    selected_nodes = set().union(*medium_components)
    G_selected = G.subgraph(selected_nodes)

    # Plot the combined subgraph
    plt.figure(figsize=(12, 8))
    pos = nx.spring_layout(G_selected, seed=42)  # Layout for visualization
    nx.draw(G_selected, pos, node_size=10, edge_color="gray", alpha=0.6)

    # Title and Save
    plt.title("Visualization of All Medium-Sized Components (100-200 Nodes)")
    plt.savefig("All_Medium_Subgraphs_100to200.png")  # Save the image
    plt.show()
else:
    print("No components found within the 100-200 node range.")
```



In [ ]:
```
clustering_coeffs = list(nx.clustering(G_removed).values())

plt.hist(clustering_coeffs, bins=50, color='green', alpha=0.7)
plt.xlabel("Clustering Coefficient")
plt.ylabel("Number of Nodes")
plt.title("Clustering Coefficient Distribution")

# Save the figure as a high-quality image
plt.savefig("Clustering Coefficient Distribution.png")

plt.show()
```



removing ordinary nodes

In [ ]:
```
# Remove random 10% of nodes
import random
num_remove = int(0.1 * len(G.nodes))  # 10% of nodes
remove_nodes = random.sample(list(G.nodes), num_remove)

G_removed_2 = G.copy()
G_removed_2.remove_nodes_from(remove_nodes)

# Check impact on connectivity
components_after = list(nx.connected_components(G_removed_2))
print(f"Remaining Components after Removal: {len(components_after)}")

# Check if network is still connected
is_still_connected = nx.is_connected(G_removed_2)
print(f"After removing 10% of Highest degree nodes, network still connected? {is_st

# Find largest remaining connected component
largest_cc_after_removal = max(nx.connected_components(G_removed_2), key=len)
print(f"Largest remaining component size: {len(largest_cc_after_removal)}")

#fragmentation index
fragmentation = 1 - (len(largest_cc_after_removal) / G_removed_2.number_of_nodes())
print(f"Network fragmentation index: {fragmentation:.2f}")

#computing the average component size
avg_component_size = sum(len(c) for c in components_after) / len(components_after)
```

```
print(f"Average component size: {avg_component_size:.2f}")

#checking the network to determine the plot
print(f"Nodes: {G_removed_2.number_of_nodes()}")
print(f"Edges: {G_removed_2.number_of_edges()}")

#Computing the Network Density
density = nx.density(G_removed_2)
print(f"Network Density: {density:.6f}")

#computing Average Clustering Coefficient
clustering_coeff = nx.average_clustering(G_removed_2)
print(f"Average Clustering Coefficient: {clustering_coeff:.4f}")
```

```
Remaining Components after Removal: 3824
After removing 10% of Highest degree nodes, network still connected? False
Largest remaining component size: 292413
Network fragmentation index: 0.03
Average component size: 78.81
Nodes: 301377
Edges: 749882
Network Density: 0.000017
Average Clustering Coefficient: 0.3852
```

In [ ]:
```
# Get all connected components and their sizes
components = sorted(nx.connected_components(G_removed_2), key=len, reverse=True)

# Create a DataFrame
df_components = pd.DataFrame({
    "Component": [f"Component {i+1}" for i in range(len(components))],
    "Size": [len(comp) for comp in components]
})
```

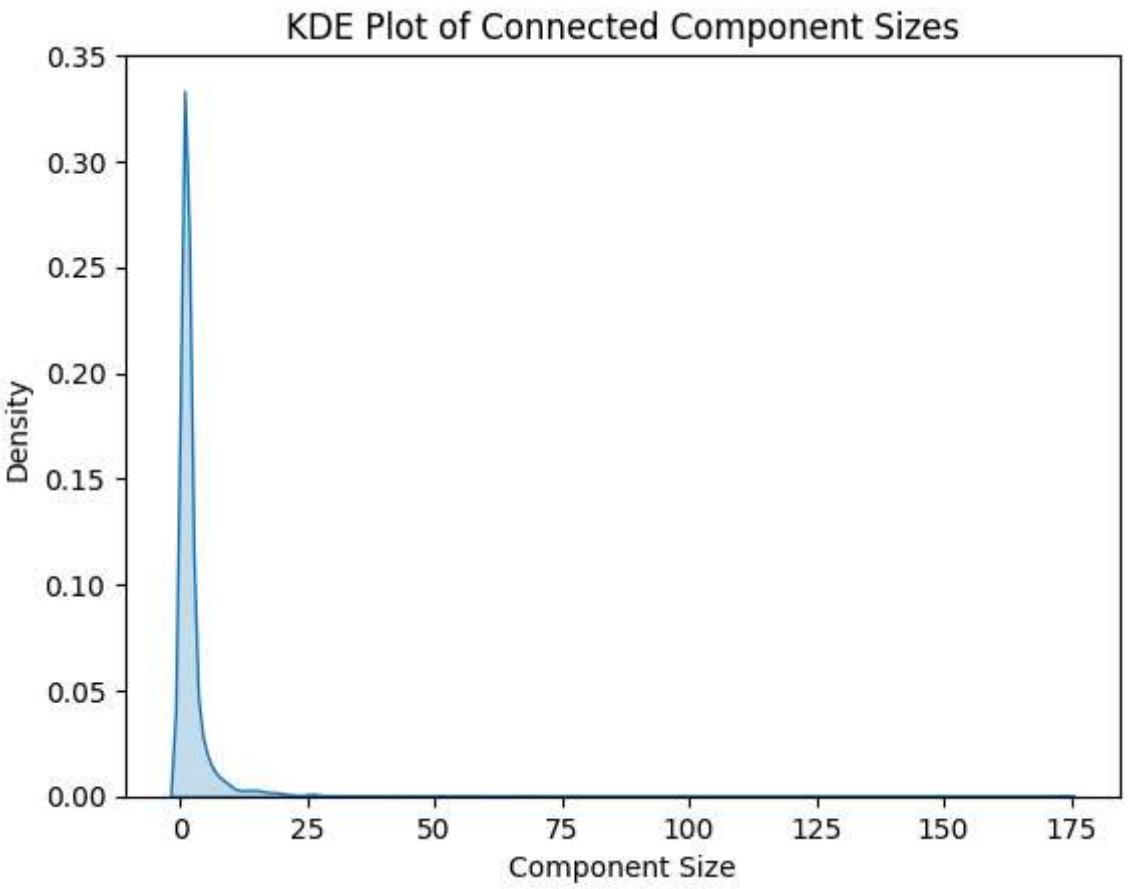
In [ ]:
```
df_components.describe()
```

Out[ ]:

	Size
count	3824.000000
mean	78.811977
std	4728.620661
min	1.000000
25%	1.000000
50%	1.000000
75%	2.000000
max	292413.000000

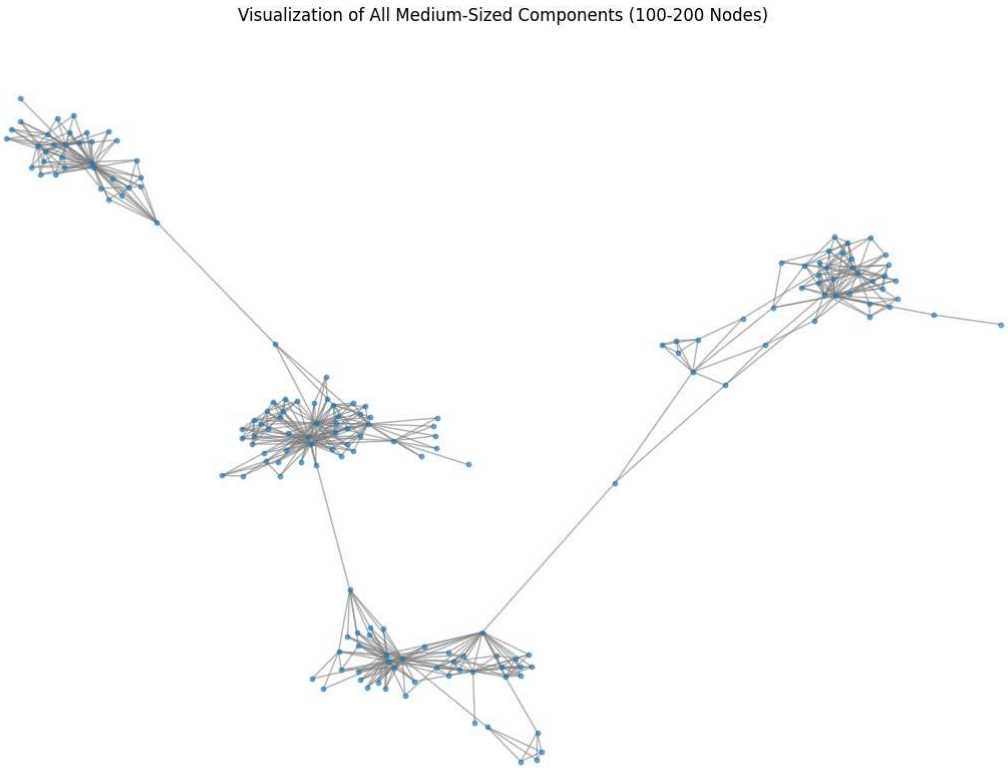
In [ ]:
```
# Create the KDE plot with fill=True for a filled area under the curve
filtered_df = df_components[df_components['Size'] < 292413]  # Example: Filter out
sns.kdeplot(filtered_df['Size'], fill=True)
```

```
# Customize the plot
plt.xlabel("Component Size")  # Set x-axis label
plt.ylabel("Density")  # Set y-axis label
plt.title("KDE Plot of Connected Component Sizes")  # Set plot title

# Display the plot
plt.show()
```

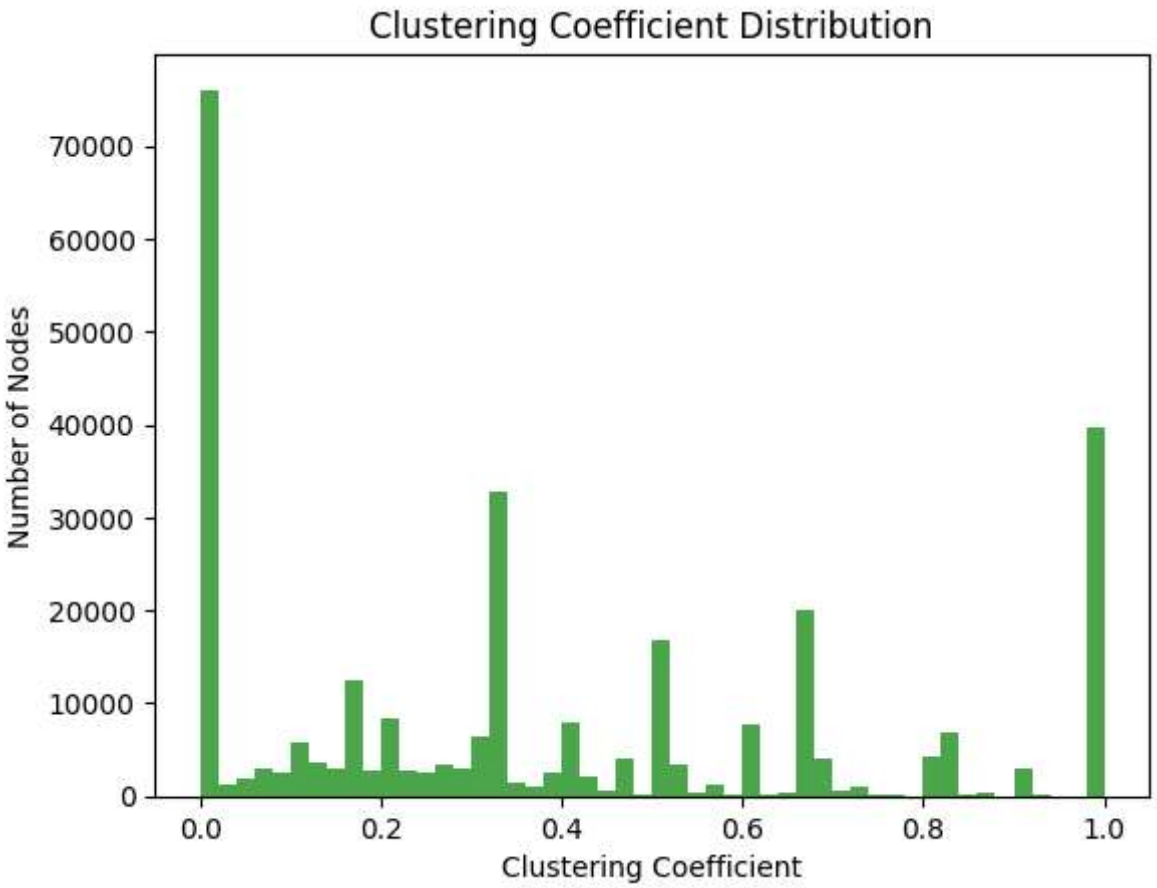


In [ ]:
```
import networkx as nx
import matplotlib.pyplot as plt

# Find all connected components with 100-200 nodes
medium_components = [comp for comp in nx.connected_components(G_removed_2) if 100 <

if medium_components:
    # Merge all selected subgraphs into one graph
    selected_nodes = set().union(*medium_components)
    G_selected = G.subgraph(selected_nodes)

    # Plot the combined subgraph
    plt.figure(figsize=(12, 8))
    pos = nx.spring_layout(G_selected, seed=42)  # Layout for visualization
    nx.draw(G_selected, pos, node_size=10, edge_color="gray", alpha=0.6)

    # Title and Save
    plt.title("Visualization of All Medium-Sized Components (100-200 Nodes)")
    plt.savefig("All_Medium_Subgraphs_100to200.png")  # Save the image
```

```
    plt.show()
else:
    print("No components found within the 100-200 node range.")
```

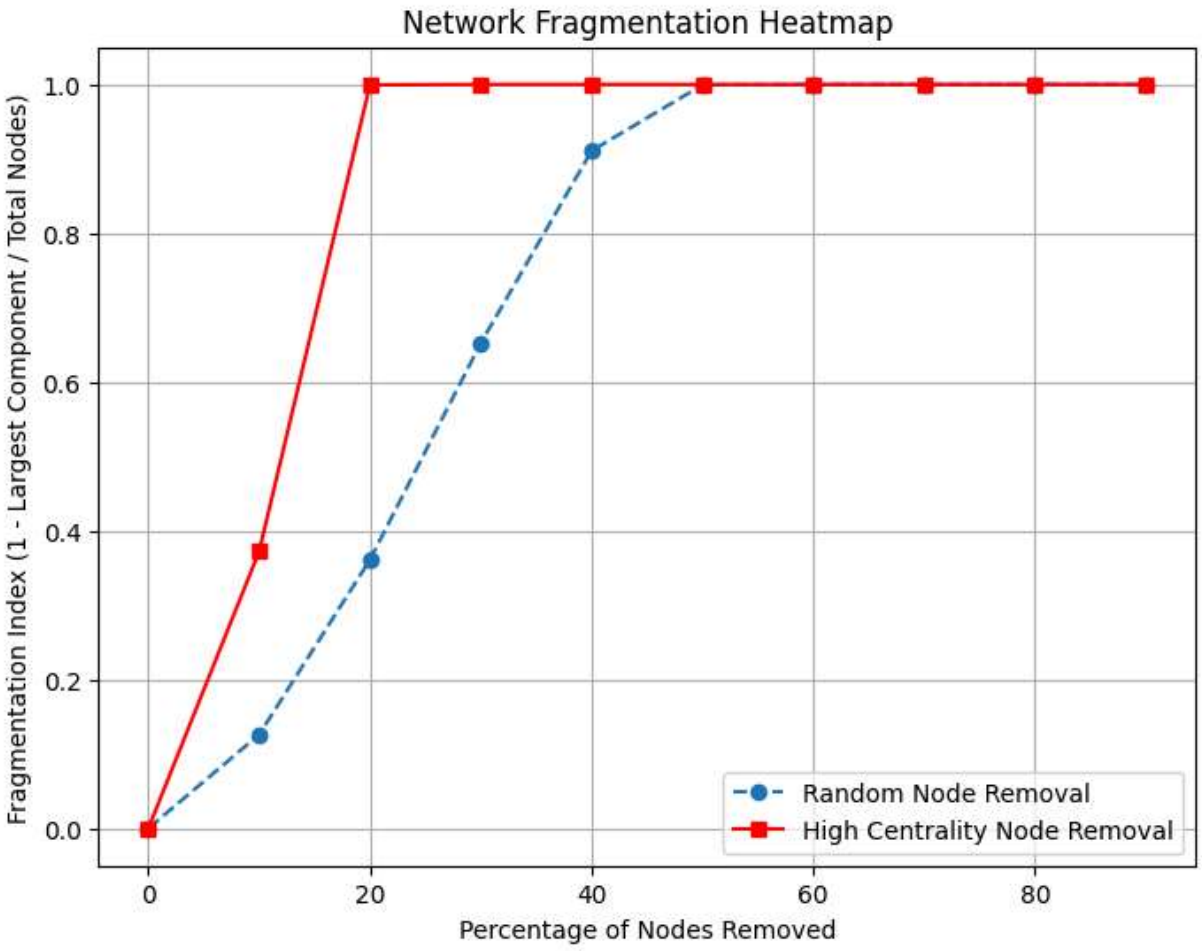


In [ ]:
```
clustering_coeffs = list(nx.clustering(G_removed_2).values())

plt.hist(clustering_coeffs, bins=50, color='green', alpha=0.7)
plt.xlabel("Clustering Coefficient")
plt.ylabel("Number of Nodes")
plt.title("Clustering Coefficient Distribution")

# Save the figure as a high-quality image
plt.savefig("Clustering Coefficient Distribution.png")

plt.show()
```



In [ ]:
```python
# Function to compute fragmentation over increasing node removals
def compute_fragmentation(G, removal_strategy='random', steps=10):
    percentages = np.linspace(0, 0.9, steps)  # Remove from 0% to 90%
    fragmentation_scores = []

    G_copy = G.copy()  # Work on a copy

    for percentage in percentages:
        num_remove = int(len(G_copy.nodes) * percentage)

        if removal_strategy == 'random':
            remove_nodes = np.random.choice(list(G_copy.nodes), num_remove, replace
        elif removal_strategy == 'high_centrality':
            centrality = nx.degree_centrality(G_copy)
            remove_nodes = sorted(centrality, key=centrality.get, reverse=True)[:nu

        G_copy.remove_nodes_from(remove_nodes)

        if len(G_copy.nodes) == 0:
            largest_cc_size = 0  # If no nodes remain
        else:
            largest_cc_size = len(max(nx.connected_components(G_copy), key=len))

        fragmentation_scores.append(1 - (largest_cc_size / len(G.nodes)))  # Fragme

    return percentages, fragmentation_scores

# Compute fragmentation for both random removal and targeted removal
```

```
random_x, random_y = compute_fragmentation(G, removal_strategy='random')
central_x, central_y = compute_fragmentation(G, removal_strategy='high_centrality')

# Plot Heatmap
plt.figure(figsize=(8, 6))
plt.plot(random_x * 100, random_y, label="Random Node Removal", marker="o", linesty
plt.plot(central_x * 100, central_y, label="High Centrality Node Removal", marker="

plt.xlabel("Percentage of Nodes Removed")
plt.ylabel("Fragmentation Index (1 - Largest Component / Total Nodes)")
plt.title("Network Fragmentation Heatmap")
plt.legend()
plt.grid()

# Save the figure as a high-quality image
plt.savefig("Network Fragmentation Heatmap.png")
plt.show()
```



RESEARCH 5: Is there a relationship between community size and density in a product co-purchase network?

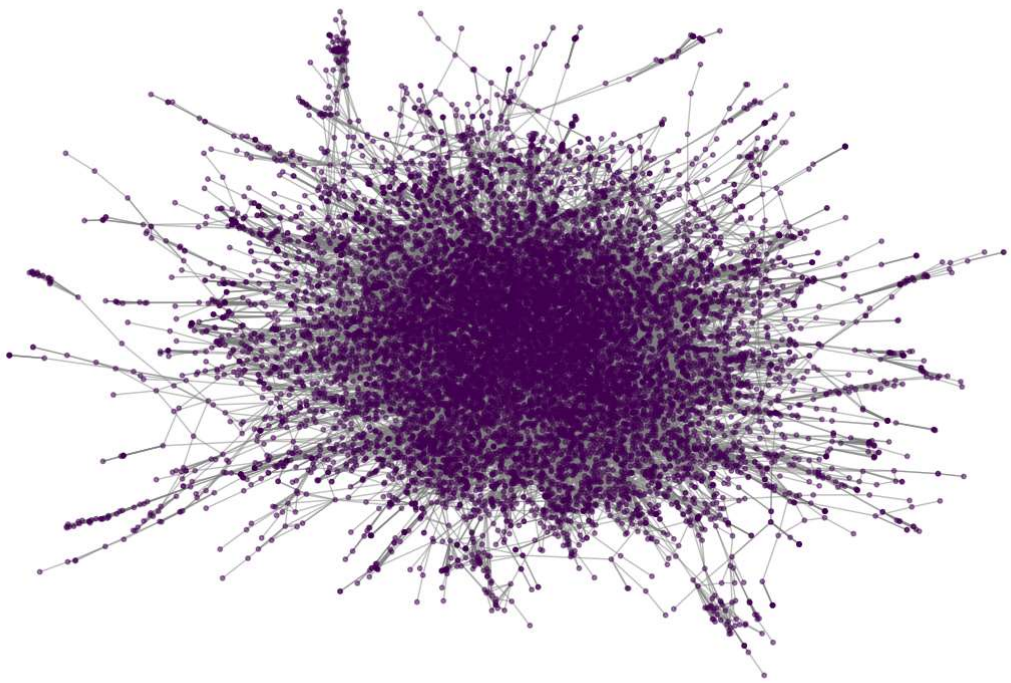
In [ ]:
```
# Create an undirected graph
G = nx.from_pandas_edgelist(df, 'FromNodeId', 'ToNodeId')
```

```
import community.community_louvain as community

# Apply Louvain method for community detection
partition = community.best_partition(G)
```

In [ ]:
```
# Count unique communities
num_communities = len(set(partition.values()))

# Group nodes by their community
from collections import defaultdict
community_groups = defaultdict(set)
for node, comm in partition.items():
    community_groups[comm].add(node)

# Sort communities by size
sorted_communities = sorted(community_groups.values(), key=len, reverse=True)

# Print community statistics
print(f"Number of communities detected: {num_communities}")
print(f"Size of largest community: {len(sorted_communities[0])}")
```

```
Number of communities detected: 252
Size of largest community: 13041
```

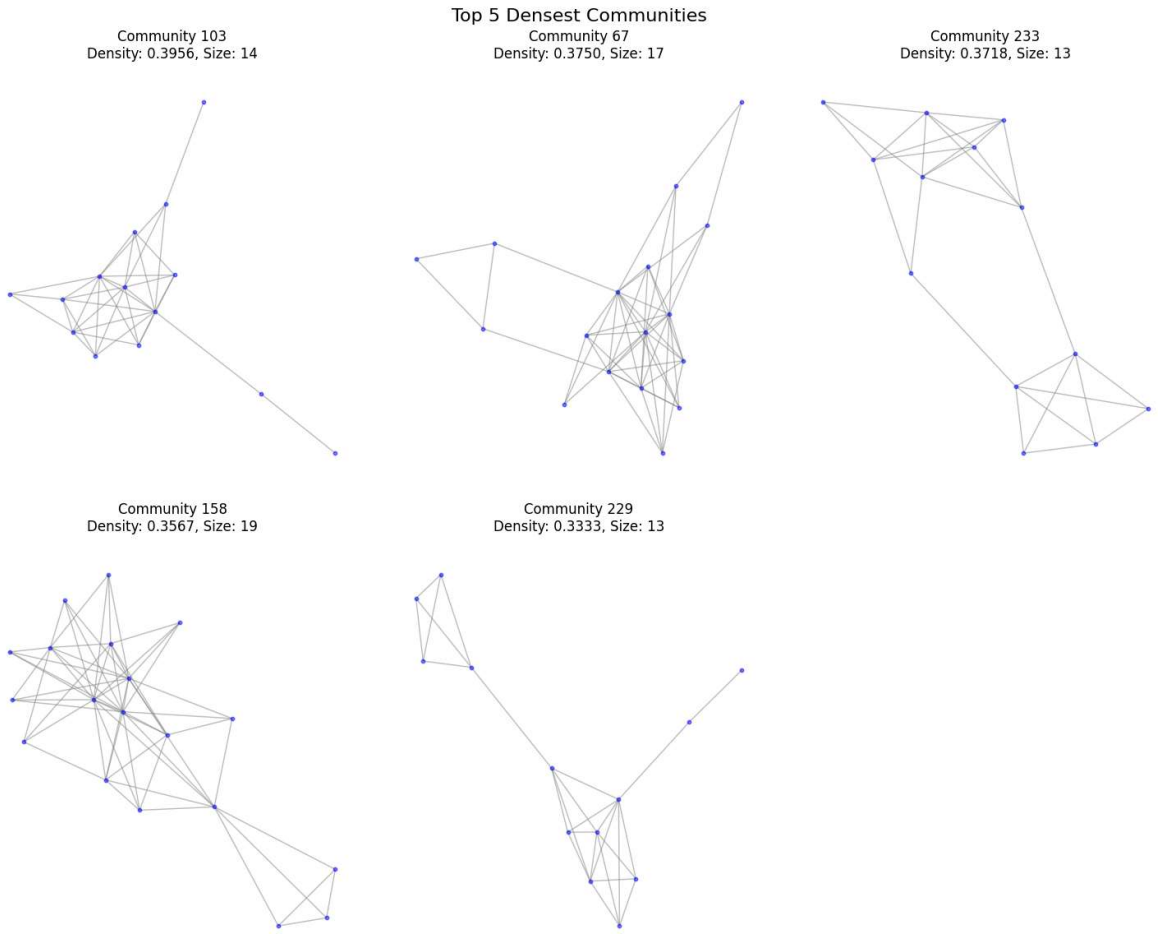
In [ ]:
```
from collections import Counter

# Count the size of each community
community_sizes = Counter(partition.values())

# Get the largest community
largest_community = community_sizes.most_common(1)[0][0]

# Filter nodes that belong to the largest community
filtered_nodes = [node for node in G.nodes() if partition[node] == largest_communit

# Create a subgraph with only the largest community
G_filtered = G.subgraph(filtered_nodes)

# Assign colors to remaining nodes (single color since it's one community)
node_colors = [partition[node] for node in G_filtered.nodes()]

# Plot the filtered graph
plt.figure(figsize=(12, 8))
pos = nx.spring_layout(G_filtered, seed=42)  # Faster than Kamada-Kawai for large g
nx.draw(G_filtered, pos, node_color=node_colors, cmap=plt.cm.viridis, node_size=10,

# Title
plt.title("Louvain Community Detection (Largest Community)")

# Save the figure as a high-quality image
plt.savefig("Louvain_Community_Detection_Largest.png", dpi=300, bbox_inches="tight"

# Show the plot
plt.show()
```



In [ ]:
```
from collections import Counter

# Count the size of each community
community_sizes = Counter(partition.values())

# Create subgraphs for each community
community_subgraphs = {
    comm: G.subgraph([node for node in G.nodes() if partition[node] == comm])
    for comm in community_sizes.keys()
}

# Compute density for each community
community_densities = {
    comm: nx.density(subgraph) for comm, subgraph in community_subgraphs.items()
}

# Sort by density (descending order)
densest_communities = sorted(community_densities.items(), key=lambda x: x[1], rever
```

In [ ]:
```
# Sort by density (descending order) and select the top 5 densest communities
top_5_densest = sorted(community_densities.items(), key=lambda x: x[1], reverse=Tru

# Plot each of the top 5 densest communities
plt.figure(figsize=(15, 12))

for i, (comm, density) in enumerate(top_5_densest, 1):
    subgraph = community_subgraphs[comm]

    plt.subplot(2, 3, i)  # Create a grid of 2 rows, 3 columns
    pos = nx.spring_layout(subgraph, seed=42)  # Position nodes using spring layout
```

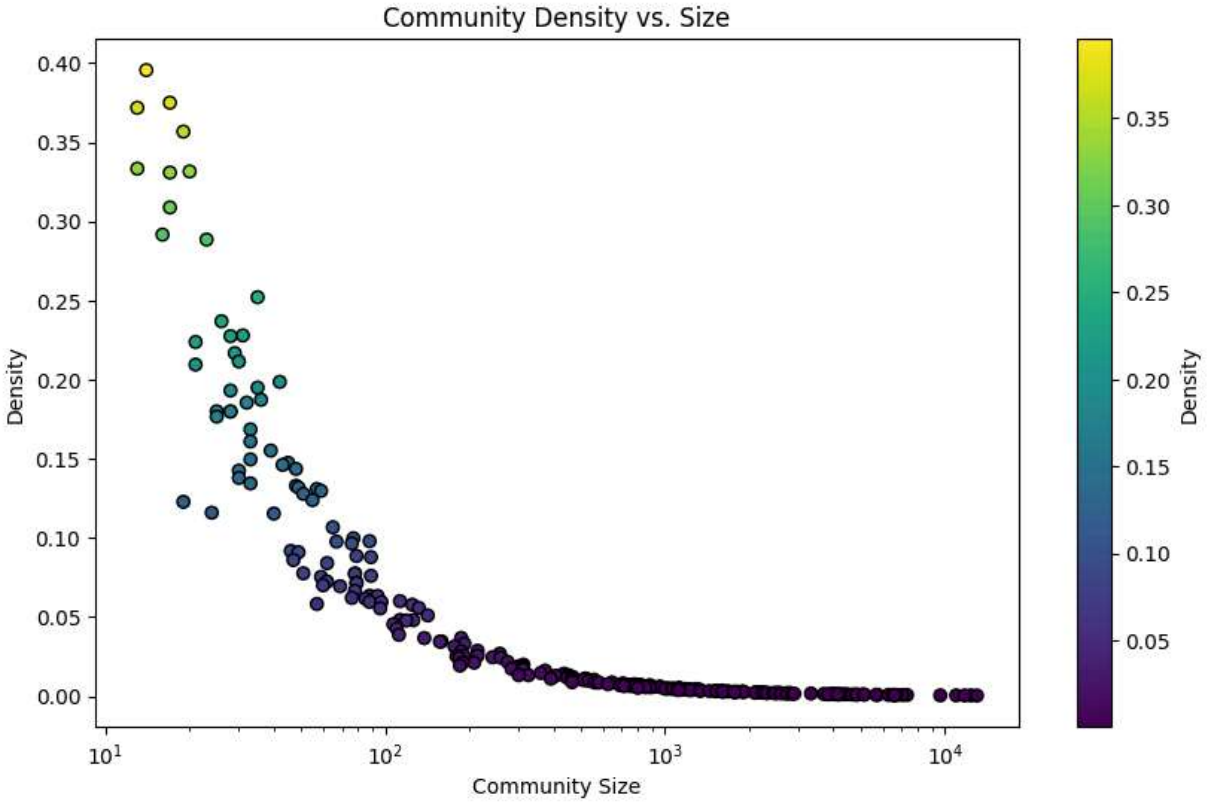
```
    nx.draw(
        subgraph, pos,
        node_color="blue", node_size=10,
        edge_color="gray", alpha=0.5
    )

    plt.title(f"Community {comm}\nDensity: {density:.4f}, Size: {len(subgraph.nodes

plt.suptitle("Top 5 Densest Communities", fontsize=16)
plt.tight_layout()


# Save the figure as a high-quality image
plt.savefig("dense_communities.png", dpi=300, bbox_inches="tight")

plt.show()
```



In [ ]:
```
# Create a DataFrame with community, density, and size
df_densest = pd.DataFrame([
    {"Community": comm, "Density": density, "Size": len(community_subgraphs[comm].n
    for comm, density in densest_communities
])
```

In [ ]:
```
plt.figure(figsize=(10, 6))
plt.scatter(df_densest["Size"], df_densest["Density"], c=df_densest["Density"], cma
```

```
plt.colorbar(label="Density")
plt.xlabel("Community Size")
plt.ylabel("Density")
plt.title("Community Density vs. Size")
plt.xscale("log")  # Log scale if sizes vary greatly
plt.show()
```



RESEARCH QUESTION SIX: HOW ACCURATE IS A MACHINE LEARNING MODEL TRAINED ON NETWORK FEATURES FOR LINK PREDICTION

In [5]:
```
!pip install torch
```

```
Requirement already satisfied: torch in /usr/local/lib/python3.11/dist-packages (2.
6.0+cu124)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages
(from torch) (3.18.0)
Requirement already satisfied: typing-extensions>=4.10.0 in /usr/local/lib/python3.1
1/dist-packages (from torch) (4.12.2)
Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-packages
(from torch) (3.4.2)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (fr
om torch) (3.1.6)
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packages (fr
om torch) (2025.3.0)
Collecting nvidia-cuda-nvrtc-cu12==12.4.127 (from torch)
  Using cached nvidia_cuda_nvrtc_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl.met
adata (1.5 kB)
Collecting nvidia-cuda-runtime-cu12==12.4.127 (from torch)
  Using cached nvidia_cuda_runtime_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl.m
etadata (1.5 kB)
Collecting nvidia-cuda-cupti-cu12==12.4.127 (from torch)
  Using cached nvidia_cuda_cupti_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl.met
adata (1.6 kB)
Collecting nvidia-cudnn-cu12==9.1.0.70 (from torch)
  Using cached nvidia_cudnn_cu12-9.1.0.70-py3-none-manylinux2014_x86_64.whl.metadata
(1.6 kB)
Collecting nvidia-cublas-cu12==12.4.5.8 (from torch)
  Using cached nvidia_cublas_cu12-12.4.5.8-py3-none-manylinux2014_x86_64.whl.metadat
a (1.5 kB)
Collecting nvidia-cufft-cu12==11.2.1.3 (from torch)
  Using cached nvidia_cufft_cu12-11.2.1.3-py3-none-manylinux2014_x86_64.whl.metadata
(1.5 kB)
Collecting nvidia-curand-cu12==10.3.5.147 (from torch)
  Using cached nvidia_curand_cu12-10.3.5.147-py3-none-manylinux2014_x86_64.whl.metad
ata (1.5 kB)
Collecting nvidia-cusolver-cu12==11.6.1.9 (from torch)
  Using cached nvidia_cusolver_cu12-11.6.1.9-py3-none-manylinux2014_x86_64.whl.metad
ata (1.6 kB)
Collecting nvidia-cusparse-cu12==12.3.1.170 (from torch)
  Using cached nvidia_cusparse_cu12-12.3.1.170-py3-none-manylinux2014_x86_64.whl.met
adata (1.6 kB)
Requirement already satisfied: nvidia-cusparselt-cu12==0.6.2 in /usr/local/lib/pytho
n3.11/dist-packages (from torch) (0.6.2)
Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in /usr/local/lib/python3.1
1/dist-packages (from torch) (2.21.5)
Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in /usr/local/lib/python3.
11/dist-packages (from torch) (12.4.127)
Collecting nvidia-nvjitlink-cu12==12.4.127 (from torch)
  Using cached nvidia_nvjitlink_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl.meta
data (1.5 kB)
Requirement already satisfied: triton==3.2.0 in /usr/local/lib/python3.11/dist-packa
ges (from torch) (3.2.0)
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.11/dist-packa
ges (from torch) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.11/dist-
packages (from sympy==1.13.1->torch) (1.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-pac
kages (from jinja2->torch) (3.0.2)
```

```
Using cached nvidia_cublas_cu12-12.4.5.8-py3-none-manylinux2014_x86_64.whl (363.4 M
B)
Using cached nvidia_cuda_cupti_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (13.8
MB)
Using cached nvidia_cuda_nvrtc_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (24.6
MB)
Using cached nvidia_cuda_runtime_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (88
3 kB)
Using cached nvidia_cudnn_cu12-9.1.0.70-py3-none-manylinux2014_x86_64.whl (664.8 MB)
Using cached nvidia_cufft_cu12-11.2.1.3-py3-none-manylinux2014_x86_64.whl (211.5 MB)
Using cached nvidia_curand_cu12-10.3.5.147-py3-none-manylinux2014_x86_64.whl (56.3 M
B)
Using cached nvidia_cusolver_cu12-11.6.1.9-py3-none-manylinux2014_x86_64.whl (127.9
MB)
Using cached nvidia_cusparse_cu12-12.3.1.170-py3-none-manylinux2014_x86_64.whl (207.
5 MB)
Using cached nvidia_nvjitlink_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (21.1
MB)
Installing collected packages: nvidia-nvjitlink-cu12, nvidia-curand-cu12, nvidia-cuf
ft-cu12, nvidia-cuda-runtime-cu12, nvidia-cuda-nvrtc-cu12, nvidia-cuda-cupti-cu12, n
vidia-cublas-cu12, nvidia-cusparse-cu12, nvidia-cudnn-cu12, nvidia-cusolver-cu12
  Attempting uninstall: nvidia-nvjitlink-cu12
    Found existing installation: nvidia-nvjitlink-cu12 12.5.82
    Uninstalling nvidia-nvjitlink-cu12-12.5.82:
      Successfully uninstalled nvidia-nvjitlink-cu12-12.5.82
  Attempting uninstall: nvidia-curand-cu12
    Found existing installation: nvidia-curand-cu12 10.3.6.82
    Uninstalling nvidia-curand-cu12-10.3.6.82:
      Successfully uninstalled nvidia-curand-cu12-10.3.6.82
  Attempting uninstall: nvidia-cufft-cu12
    Found existing installation: nvidia-cufft-cu12 11.2.3.61
    Uninstalling nvidia-cufft-cu12-11.2.3.61:
      Successfully uninstalled nvidia-cufft-cu12-11.2.3.61
  Attempting uninstall: nvidia-cuda-runtime-cu12
    Found existing installation: nvidia-cuda-runtime-cu12 12.5.82
    Uninstalling nvidia-cuda-runtime-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-runtime-cu12-12.5.82
  Attempting uninstall: nvidia-cuda-nvrtc-cu12
    Found existing installation: nvidia-cuda-nvrtc-cu12 12.5.82
    Uninstalling nvidia-cuda-nvrtc-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-nvrtc-cu12-12.5.82
  Attempting uninstall: nvidia-cuda-cupti-cu12
    Found existing installation: nvidia-cuda-cupti-cu12 12.5.82
    Uninstalling nvidia-cuda-cupti-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-cupti-cu12-12.5.82
  Attempting uninstall: nvidia-cublas-cu12
    Found existing installation: nvidia-cublas-cu12 12.5.3.2
    Uninstalling nvidia-cublas-cu12-12.5.3.2:
      Successfully uninstalled nvidia-cublas-cu12-12.5.3.2
  Attempting uninstall: nvidia-cusparse-cu12
    Found existing installation: nvidia-cusparse-cu12 12.5.1.3
    Uninstalling nvidia-cusparse-cu12-12.5.1.3:
      Successfully uninstalled nvidia-cusparse-cu12-12.5.1.3
  Attempting uninstall: nvidia-cudnn-cu12
    Found existing installation: nvidia-cudnn-cu12 9.3.0.75
    Uninstalling nvidia-cudnn-cu12-9.3.0.75:
```

```
    Successfully uninstalled nvidia-cudnn-cu12-9.3.0.75
  Attempting uninstall: nvidia-cusolver-cu12
    Found existing installation: nvidia-cusolver-cu12 11.6.3.83
    Uninstalling nvidia-cusolver-cu12-11.6.3.83:
      Successfully uninstalled nvidia-cusolver-cu12-11.6.3.83
Successfully installed nvidia-cublas-cu12-12.4.5.8 nvidia-cuda-cupti-cu12-12.4.127 n
vidia-cuda-nvrtc-cu12-12.4.127 nvidia-cuda-runtime-cu12-12.4.127 nvidia-cudnn-cu12-
9.1.0.70 nvidia-cufft-cu12-11.2.1.3 nvidia-curand-cu12-10.3.5.147 nvidia-cusolver-cu
12-11.6.1.9 nvidia-cusparse-cu12-12.3.1.170 nvidia-nvjitlink-cu12-12.4.127
```

In [6]:
```
!pip install torchvision
```

```
Requirement already satisfied: torchvision in /usr/local/lib/python3.11/dist-package
s (0.21.0+cu124)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (fro
m torchvision) (2.0.2)
Requirement already satisfied: torch==2.6.0 in /usr/local/lib/python3.11/dist-packag
es (from torchvision) (2.6.0+cu124)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.11/di
st-packages (from torchvision) (11.1.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages
(from torch==2.6.0->torchvision) (3.18.0)
Requirement already satisfied: typing-extensions>=4.10.0 in /usr/local/lib/python3.1
1/dist-packages (from torch==2.6.0->torchvision) (4.12.2)
Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-packages
(from torch==2.6.0->torchvision) (3.4.2)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (fr
om torch==2.6.0->torchvision) (3.1.6)
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packages (fr
om torch==2.6.0->torchvision) (2025.3.0)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.4.127 in /usr/local/lib/py
thon3.11/dist-packages (from torch==2.6.0->torchvision) (12.4.127)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.4.127 in /usr/local/lib/
python3.11/dist-packages (from torch==2.6.0->torchvision) (12.4.127)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.4.127 in /usr/local/lib/py
thon3.11/dist-packages (from torch==2.6.0->torchvision) (12.4.127)
Requirement already satisfied: nvidia-cudnn-cu12==9.1.0.70 in /usr/local/lib/python
3.11/dist-packages (from torch==2.6.0->torchvision) (9.1.0.70)
Requirement already satisfied: nvidia-cublas-cu12==12.4.5.8 in /usr/local/lib/python
3.11/dist-packages (from torch==2.6.0->torchvision) (12.4.5.8)
Requirement already satisfied: nvidia-cufft-cu12==11.2.1.3 in /usr/local/lib/python
3.11/dist-packages (from torch==2.6.0->torchvision) (11.2.1.3)
Requirement already satisfied: nvidia-curand-cu12==10.3.5.147 in /usr/local/lib/pyth
on3.11/dist-packages (from torch==2.6.0->torchvision) (10.3.5.147)
Requirement already satisfied: nvidia-cusolver-cu12==11.6.1.9 in /usr/local/lib/pyth
on3.11/dist-packages (from torch==2.6.0->torchvision) (11.6.1.9)
Requirement already satisfied: nvidia-cusparse-cu12==12.3.1.170 in /usr/local/lib/py
thon3.11/dist-packages (from torch==2.6.0->torchvision) (12.3.1.170)
Requirement already satisfied: nvidia-cusparselt-cu12==0.6.2 in /usr/local/lib/pytho
n3.11/dist-packages (from torch==2.6.0->torchvision) (0.6.2)
Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in /usr/local/lib/python3.1
1/dist-packages (from torch==2.6.0->torchvision) (2.21.5)
Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in /usr/local/lib/python3.
11/dist-packages (from torch==2.6.0->torchvision) (12.4.127)
Requirement already satisfied: nvidia-nvjitlink-cu12==12.4.127 in /usr/local/lib/pyt
hon3.11/dist-packages (from torch==2.6.0->torchvision) (12.4.127)
Requirement already satisfied: triton==3.2.0 in /usr/local/lib/python3.11/dist-packa
ges (from torch==2.6.0->torchvision) (3.2.0)
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.11/dist-packa
ges (from torch==2.6.0->torchvision) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.11/dist-
packages (from sympy==1.13.1->torch==2.6.0->torchvision) (1.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-pac
kages (from jinja2->torch==2.6.0->torchvision) (3.0.2)
```

In [7]:
```
!pip install torchaudio
```

```
Requirement already satisfied: torchaudio in /usr/local/lib/python3.11/dist-packages
(2.6.0+cu124)
Requirement already satisfied: torch==2.6.0 in /usr/local/lib/python3.11/dist-packag
es (from torchaudio) (2.6.0+cu124)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages
(from torch==2.6.0->torchaudio) (3.18.0)
Requirement already satisfied: typing-extensions>=4.10.0 in /usr/local/lib/python3.1
1/dist-packages (from torch==2.6.0->torchaudio) (4.12.2)
Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-packages
(from torch==2.6.0->torchaudio) (3.4.2)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (fr
om torch==2.6.0->torchaudio) (3.1.6)
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packages (fr
om torch==2.6.0->torchaudio) (2025.3.0)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.4.127 in /usr/local/lib/py
thon3.11/dist-packages (from torch==2.6.0->torchaudio) (12.4.127)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.4.127 in /usr/local/lib/
python3.11/dist-packages (from torch==2.6.0->torchaudio) (12.4.127)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.4.127 in /usr/local/lib/py
thon3.11/dist-packages (from torch==2.6.0->torchaudio) (12.4.127)
Requirement already satisfied: nvidia-cudnn-cu12==9.1.0.70 in /usr/local/lib/python
3.11/dist-packages (from torch==2.6.0->torchaudio) (9.1.0.70)
Requirement already satisfied: nvidia-cublas-cu12==12.4.5.8 in /usr/local/lib/python
3.11/dist-packages (from torch==2.6.0->torchaudio) (12.4.5.8)
Requirement already satisfied: nvidia-cufft-cu12==11.2.1.3 in /usr/local/lib/python
3.11/dist-packages (from torch==2.6.0->torchaudio) (11.2.1.3)
Requirement already satisfied: nvidia-curand-cu12==10.3.5.147 in /usr/local/lib/pyth
on3.11/dist-packages (from torch==2.6.0->torchaudio) (10.3.5.147)
Requirement already satisfied: nvidia-cusolver-cu12==11.6.1.9 in /usr/local/lib/pyth
on3.11/dist-packages (from torch==2.6.0->torchaudio) (11.6.1.9)
Requirement already satisfied: nvidia-cusparse-cu12==12.3.1.170 in /usr/local/lib/py
thon3.11/dist-packages (from torch==2.6.0->torchaudio) (12.3.1.170)
Requirement already satisfied: nvidia-cusparselt-cu12==0.6.2 in /usr/local/lib/pytho
n3.11/dist-packages (from torch==2.6.0->torchaudio) (0.6.2)
Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in /usr/local/lib/python3.1
1/dist-packages (from torch==2.6.0->torchaudio) (2.21.5)
Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in /usr/local/lib/python3.
11/dist-packages (from torch==2.6.0->torchaudio) (12.4.127)
Requirement already satisfied: nvidia-nvjitlink-cu12==12.4.127 in /usr/local/lib/pyt
hon3.11/dist-packages (from torch==2.6.0->torchaudio) (12.4.127)
Requirement already satisfied: triton==3.2.0 in /usr/local/lib/python3.11/dist-packa
ges (from torch==2.6.0->torchaudio) (3.2.0)
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.11/dist-packa
ges (from torch==2.6.0->torchaudio) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.11/dist-
packages (from sympy==1.13.1->torch==2.6.0->torchaudio) (1.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-pac
kages (from jinja2->torch==2.6.0->torchaudio) (3.0.2)
```

In [8]:
```
!pip install torch-geometric
```

```
Collecting torch-geometric
  Using cached torch_geometric-2.6.1-py3-none-any.whl.metadata (63 kB)
Requirement already satisfied: aiohttp in /usr/local/lib/python3.11/dist-packages (f
rom torch-geometric) (3.11.14)
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packages (fr
om torch-geometric) (2025.3.0)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (fr
om torch-geometric) (3.1.6)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (fro
m torch-geometric) (2.0.2)
Requirement already satisfied: psutil>=5.8.0 in /usr/local/lib/python3.11/dist-packa
ges (from torch-geometric) (5.9.5)
Requirement already satisfied: pyparsing in /usr/local/lib/python3.11/dist-packages
(from torch-geometric) (3.2.1)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages
(from torch-geometric) (2.32.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from
torch-geometric) (4.67.1)
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in /usr/local/lib/python3.11/
dist-packages (from aiohttp->torch-geometric) (2.6.1)
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.11/dist-pa
ckages (from aiohttp->torch-geometric) (1.3.2)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.11/dist-packa
ges (from aiohttp->torch-geometric) (25.3.0)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.11/dist-p
ackages (from aiohttp->torch-geometric) (1.5.0)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.11/dist
-packages (from aiohttp->torch-geometric) (6.2.0)
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.11/dist-pa
ckages (from aiohttp->torch-geometric) (0.3.0)
Requirement already satisfied: yarl<2.0,>=1.17.0 in /usr/local/lib/python3.11/dist-p
ackages (from aiohttp->torch-geometric) (1.18.3)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-pac
kages (from jinja2->torch-geometric) (3.0.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.1
1/dist-packages (from requests->torch-geometric) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packag
es (from requests->torch-geometric) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-
packages (from requests->torch-geometric) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-
packages (from requests->torch-geometric) (2025.1.31)
Using cached torch_geometric-2.6.1-py3-none-any.whl (1.1 MB)
Installing collected packages: torch-geometric
Successfully installed torch-geometric-2.6.1
```

In [9]:
```
!pip install torch-scatter
```

```
Collecting torch-scatter
  Using cached torch_scatter-2.1.2-cp311-cp311-linux_x86_64.whl
Installing collected packages: torch-scatter
Successfully installed torch-scatter-2.1.2
```

In [14]:
```
!pip install torch_geometric
```

```
Requirement already satisfied: torch_geometric in /usr/local/lib/python3.11/dist-pac
kages (2.6.1)
Requirement already satisfied: aiohttp in /usr/local/lib/python3.11/dist-packages (f
rom torch_geometric) (3.11.14)
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packages (fr
om torch_geometric) (2025.3.0)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (fr
om torch_geometric) (3.1.6)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (fro
m torch_geometric) (2.0.2)
Requirement already satisfied: psutil>=5.8.0 in /usr/local/lib/python3.11/dist-packa
ges (from torch_geometric) (5.9.5)
Requirement already satisfied: pyparsing in /usr/local/lib/python3.11/dist-packages
(from torch_geometric) (3.2.1)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages
(from torch_geometric) (2.32.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from
torch_geometric) (4.67.1)
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in /usr/local/lib/python3.11/
dist-packages (from aiohttp->torch_geometric) (2.6.1)
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.11/dist-pa
ckages (from aiohttp->torch_geometric) (1.3.2)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.11/dist-packa
ges (from aiohttp->torch_geometric) (25.3.0)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.11/dist-p
ackages (from aiohttp->torch_geometric) (1.5.0)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.11/dist
-packages (from aiohttp->torch_geometric) (6.2.0)
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.11/dist-pa
ckages (from aiohttp->torch_geometric) (0.3.0)
Requirement already satisfied: yarl<2.0,>=1.17.0 in /usr/local/lib/python3.11/dist-p
ackages (from aiohttp->torch_geometric) (1.18.3)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-pac
kages (from jinja2->torch_geometric) (3.0.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.1
1/dist-packages (from requests->torch_geometric) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packag
es (from requests->torch_geometric) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-
packages (from requests->torch_geometric) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-
packages (from requests->torch_geometric) (2025.1.31)
```

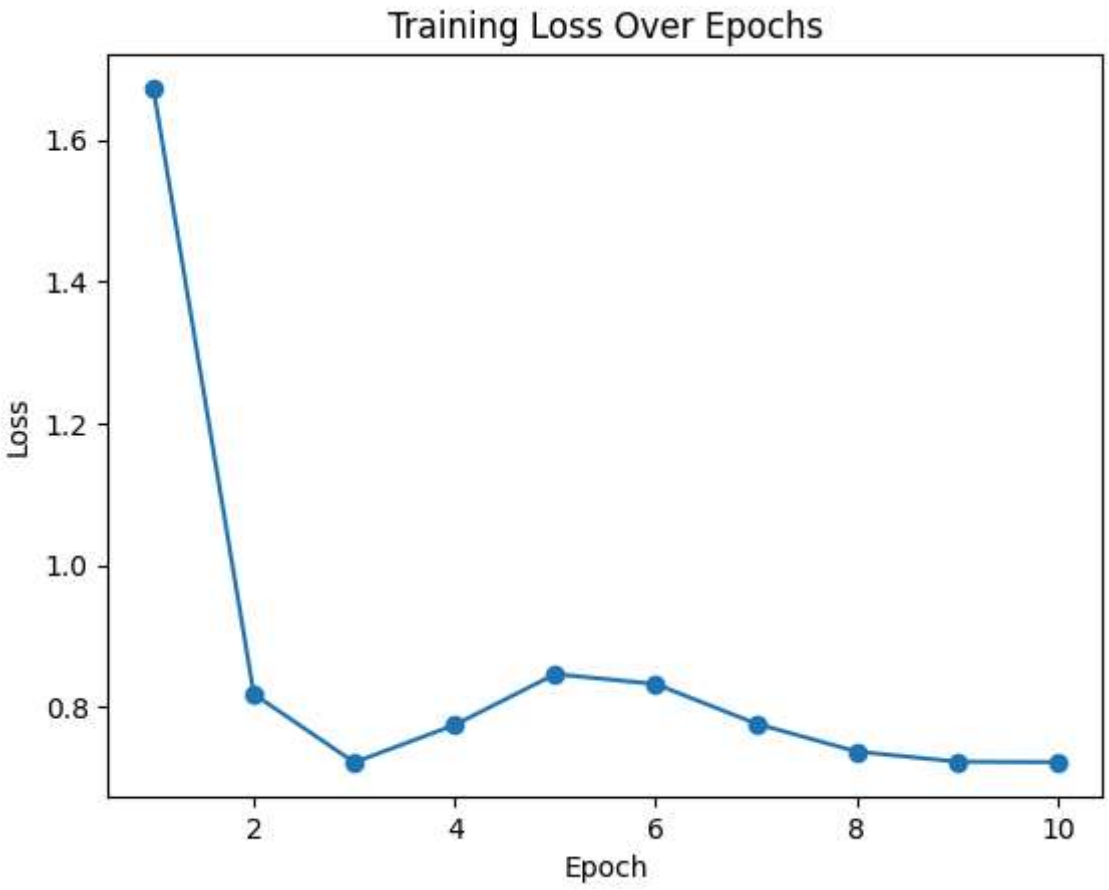
In [20]:

```python
import torch
import networkx as nx
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import torch.nn.functional as F
import torch_geometric.transforms as T
from torch_geometric.data import Data
from torch_geometric.nn import SAGEConv
from torch_geometric.utils import from_networkx, negative_sampling

# Step 1: Load Graph from Pandas Edgelist
G_full = nx.from_pandas_edgelist(df, 'FromNodeId', 'ToNodeId')
```

```
# Step 2: Compute Degree Centrality
degree_dict = dict(G_full.degree())  # Get node degrees
sorted_nodes = sorted(degree_dict.items(), key=lambda x: x[1], reverse=True)

# Step 3: Extract Top 20,000 High-Degree Nodes
high_degree_nodes = [node for node, _ in sorted_nodes[:20000]]

# Step 4: Create Induced Subgraph
G = G_full.subgraph(high_degree_nodes).copy()

# Step 5: Convert to PyG Data Format
data = from_networkx(G)

# Step 6: Add Node Features (if missing)
if not hasattr(data, 'x') or data.x is None:
    data.x = torch.ones((data.num_nodes, 1))  # Dummy feature

# Step 7: Split Data for Link Prediction
transform = T.RandomLinkSplit(num_val=0.05, num_test=0.1, is_undirected=True)
train_data, val_data, test_data = transform(data)

# Step 8: Define GraphSAGE Model
class GraphSAGEModel(torch.nn.Module):
    def __init__(self, in_channels, hidden_channels, out_channels):
        super(GraphSAGEModel, self).__init__()
        self.conv1 = SAGEConv(in_channels, hidden_channels)
        self.conv2 = SAGEConv(hidden_channels, out_channels)

    def forward(self, x, edge_index):
        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = self.conv2(x, edge_index)
        return x

# Initialize Model
in_channels = train_data.x.shape[1]
model = GraphSAGEModel(in_channels=in_channels, hidden_channels=32, out_channels=16
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)

# Training Loop
losses = []

def train():
    model.train()
    optimizer.zero_grad()
    z = model(train_data.x, train_data.edge_index)  # Get node embeddings

    # Retrieve positive and negative edge indices
    edge_index = train_data.edge_label_index
    edge_labels = train_data.edge_label.float()  # Convert to float for BCE loss

    # Compute link prediction scores
    link_logits = (z[edge_index[0]] * z[edge_index[1]]).sum(dim=1)

    # Compute binary cross-entropy loss
```

```
    loss = F.binary_cross_entropy_with_logits(link_logits, edge_labels)

    loss.backward()
    optimizer.step()
    return loss.item()

# Train for 10 Epochs
for epoch in range(10):
    loss = train()
    losses.append(loss)
    print(f"Epoch {epoch+1}, Loss: {loss:.4f}")

# Plot Training Loss Curve
plt.plot(range(1, len(losses)+1), losses, marker='o', linestyle='-')
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.title("Training Loss Over Epochs")
plt.show()

# Step 9: Evaluate Model
model.eval()
z = model(test_data.x, test_data.edge_index)

# Sample Negative Edges for Testing
test_pos_edge_index = test_data.edge_label_index  # Corrected
test_neg_edge_index = negative_sampling(
    edge_index=test_pos_edge_index,  # Use label index for proper sampling
    num_nodes=test_data.num_nodes,
    num_neg_samples=test_pos_edge_index.shape[1]
)

# Compute Test Loss
test_loss = F.binary_cross_entropy_with_logits(
    (z[test_neg_edge_index[0]] * z[test_neg_edge_index[1]]).sum(dim=1),
    torch.zeros(test_neg_edge_index.shape[1])
)
print(f"Test Loss: {test_loss:.4f}")
```

```
Epoch 1, Loss: 1.6724
Epoch 2, Loss: 0.8182
Epoch 3, Loss: 0.7218
Epoch 4, Loss: 0.7746
Epoch 5, Loss: 0.8463
Epoch 6, Loss: 0.8323
Epoch 7, Loss: 0.7764
Epoch 8, Loss: 0.7371
Epoch 9, Loss: 0.7228
Epoch 10, Loss: 0.7222
```




```
Test Loss: 1.0645
```

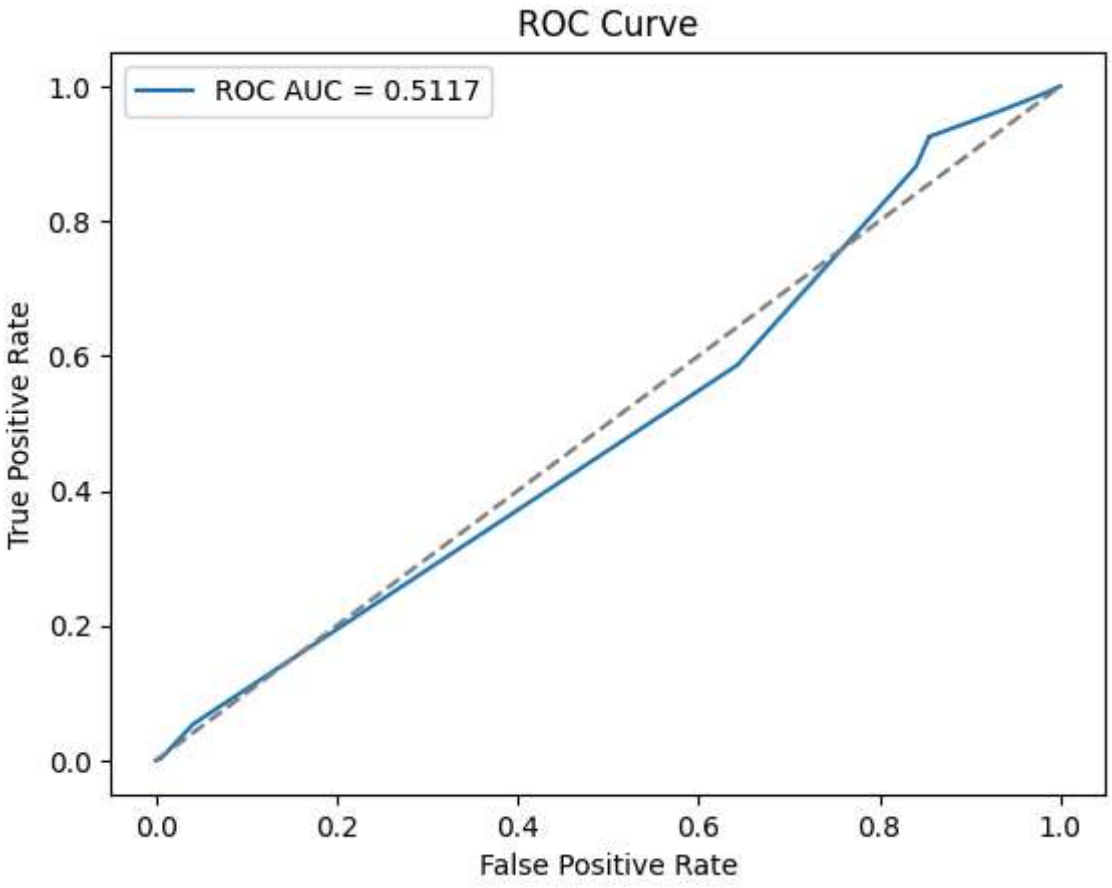
In [21]:
```python
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score, average_precision_score

def evaluate(model, data, threshold=0.0):
    model.eval()
    with torch.no_grad():
        z = model(data.x, data.edge_index)  # Get node embeddings

    # Retrieve positive and negative edges
    pos_edge_index = data.edge_label_index
    neg_edge_index = negative_sampling(
        edge_index=data.edge_index,
        num_nodes=data.num_nodes,
        num_neg_samples=pos_edge_index.shape[1]
    )

    # Compute link logits (dot product of node embeddings)
    pos_scores = (z[pos_edge_index[0]] * z[pos_edge_index[1]]).sum(dim=1)
    neg_scores = (z[neg_edge_index[0]] * z[neg_edge_index[1]]).sum(dim=1)

    # Create labels: 1 for positive edges, 0 for negative edges
    y_true = torch.cat([torch.ones(pos_scores.shape[0]), torch.zeros(neg_scores.sha
    y_scores = torch.cat([pos_scores, neg_scores]).cpu().numpy()

    # Convert scores to binary predictions using threshold (default=0.0)
    y_pred = (y_scores > threshold).astype(int)
```

```
    # Compute Accuracy, ROC AUC, and AP Score
    accuracy = accuracy_score(y_true.numpy(), y_pred)
    roc_auc = roc_auc_score(y_true.numpy(), y_scores)
    ap_score = average_precision_score(y_true.numpy(), y_scores)

    return accuracy, roc_auc, ap_score

# Evaluate on Test Data
accuracy, roc_auc, ap_score = evaluate(model, test_data)
print(f"Test Accuracy: {accuracy:.4f}, ROC AUC: {roc_auc:.4f}, Average Precision: {
```

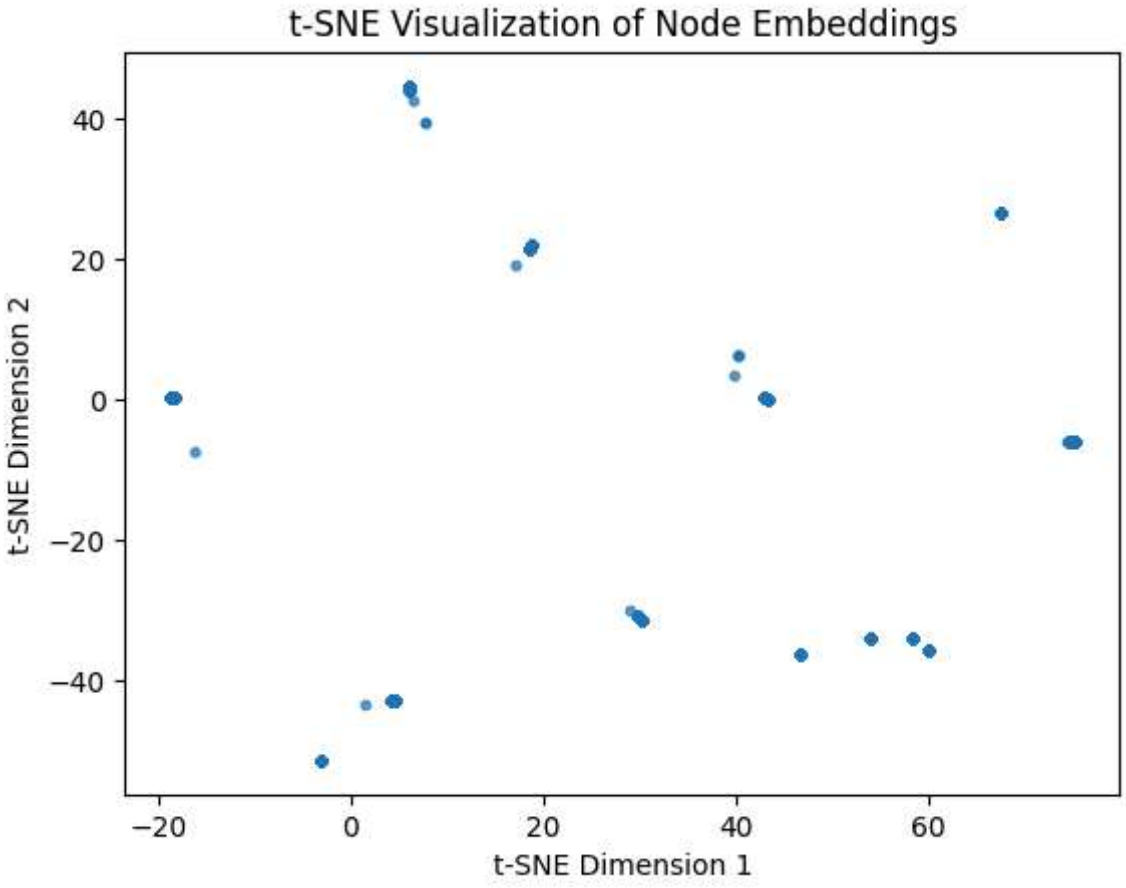
```
Test Accuracy: 0.5000, ROC AUC: 0.5117, Average Precision: 0.5088
```

In [22]:
```
from sklearn.metrics import roc_curve

model.eval()
with torch.no_grad():
  z = model(data.x, data.edge_index)

# Retrieve positive and negative edges
pos_edge_index = test_data.edge_label_index
neg_edge_index = negative_sampling(
    edge_index=test_data.edge_index,
    num_nodes=test_data.num_nodes,
    num_neg_samples=pos_edge_index.shape[1]
)

# Compute link logits (dot product of node embeddings)
pos_scores = (z[pos_edge_index[0]] * z[pos_edge_index[1]]).sum(dim=1)
neg_scores = (z[neg_edge_index[0]] * z[neg_edge_index[1]]).sum(dim=1)

# Get True Labels and Predicted Scores
y_true = torch.cat([torch.ones(pos_scores.shape[0]), torch.zeros(neg_scores.shape[0
y_scores = torch.cat([pos_scores, neg_scores]).cpu().numpy()

# Compute ROC Curve
fpr, tpr, _ = roc_curve(y_true, y_scores)

# Plot ROC Curve
plt.plot(fpr, tpr, label=f"ROC AUC = {roc_auc:.4f}")
plt.plot([0, 1], [0, 1], linestyle="--", color="gray")  # Random guessing line
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
```

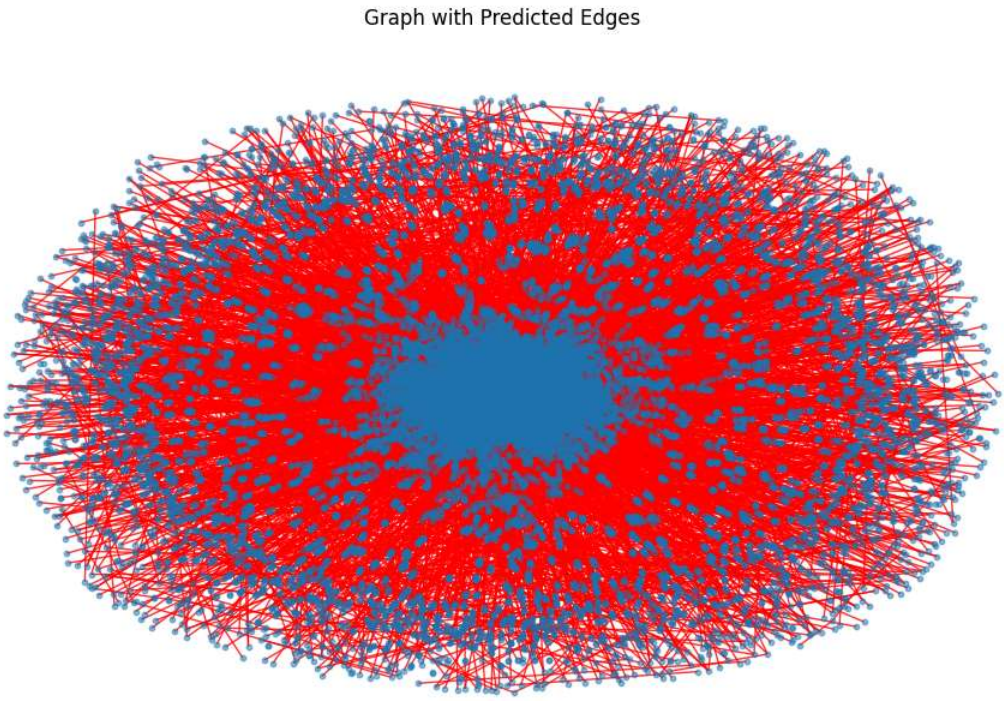
Out[22]: <matplotlib.legend.Legend at 0x7e662d3974d0>



In [23]:
```
from sklearn.manifold import TSNE

# Compute Node Embeddings
z = model(test_data.x, test_data.edge_index).cpu().detach().numpy()

# Reduce Embedding Dimensionality
z_2d = TSNE(n_components=2).fit_transform(z)

# Plot Embeddings
plt.scatter(z_2d[:, 0], z_2d[:, 1], s=10, alpha=0.7)
plt.xlabel("t-SNE Dimension 1")
plt.ylabel("t-SNE Dimension 2")
plt.title("t-SNE Visualization of Node Embeddings")
plt.show()
```



In [25]:
```
# Ensure all nodes from predicted edges exist in G
G.add_nodes_from(test_data.edge_label_index.view(-1).cpu().numpy())

# Generate layout and draw as before
pos = nx.spring_layout(G)
plt.figure(figsize=(10, 6))
nx.draw(G, pos, node_size=10, edge_color='gray', alpha=0.5)
nx.draw_networkx_edges(G, pos, edgelist=pred_edges, edge_color='red', alpha=0.8)

plt.title("Graph with Predicted Edges")
plt.show()
```



In [ ]: