GitHub Collaboration Network Analysis

Set Up Data Collection

Setup

```
# Importing Libraries
import networkx as nx
import matplotlib.pyplot as plt
import requests
import pandas as pd
import numpy as np
from collections import defaultdict
import time
import json
from IPython.display import display, Markdown, HTML
from datetime import datetime, timedelta
from google.colab import drive
```

```
# Configuration with verbose output
print("🚀 Initializing GitHub Collaboration Network Analyzer...")
plt.style.use('ggplot')
pd.set_option('display.max_colwidth', None)
print("✓ Configured display settings and plot styles")
```

```
🚀 Initializing GitHub Collaboration Network Analyzer...
✓ Configured display settings and plot styles
```

```
# GitHub API Setup with enhanced error handling
print("\n📝 Setting up GitHub API access...")
github_token = input("Enter GitHub token (press Enter for public access): ").strip()
headers = {'Accept': 'application/vnd.github.v3+json'}
if github_token:
    headers['Authorization'] = f'token {github_token}'
    print("✓ Authentication configured with provided token")

    # Test token validity
    test_response = requests.get("https://api.github.com/rate_limit", headers=headers)
    if test_response.status_code == 200:
        rate_info = test_response.json()['rate']
        print(f"✓ Token validated. Rate limit: {rate_info['remaining']}/{rate_info['limit
    else:
        print(f"⚠️ Token may not be valid. Status: {test_response.status_code}")
else:
    print("✓ Using public API access (rate limited to 60 requests/hour)")
    test_response = requests.get("https://api.github.com/rate_limit")
    if test_response.status_code == 200:
        rate_info = test_response.json()['rate']
```

```
        print(f"ℹ️ Available rate limit: {rate_info['remaining']}/{rate_info['limit']} re
```

```
📝 Setting up GitHub API access...
Enter GitHub token (press Enter for public access): github_pat_11AYEBN7A0474xDUXQw9RR
✓ Authentication configured with provided token
✓ Token validated. Rate limit: 4916/5000 requests remaining
```

Data Collection

Repository Selection with validation

```
# @title Repository Selection with validation
print("\n📋 Configuring repository search parameters...")
topic = input("Enter GitHub topic (e.g., machine-learning): ").strip() or "machine-learni
print(f"✓ Topic set to: '{topic}'")

try:
    num_repos = int(input("Number of repositories (10-50): ") or "15")
    num_repos = max(10, min(num_repos, 50))
    print(f"✓ Will collect {num_repos} repositories")
except ValueError:
    num_repos = 15
    print(f"⚠️ Invalid input. Using default: {num_repos} repositories")

try:
    days_back = int(input("Filter repos updated in last N days (0 for all): ") or "0")
    if days_back > 0:
        date_filter = (datetime.now() - timedelta(days=days_back)).strftime("%Y-%m-%d")
        print(f"✓ Filtering by repos updated since {date_filter}")
    else:
        print("✓ No date filtering applied")
except ValueError:
    days_back = 0
    print("⚠️ Invalid input. Using no date filtering")
```

```
📋 Configuring repository search parameters...
Enter GitHub topic (e.g., machine-learning):
✓ Topic set to: 'machine-learning'
Number of repositories (10-50): 50
✓ Will collect 50 repositories
Filter repos updated in last N days (0 for all):
✓ No date filtering applied
```

Details of Data Collected

- GitHub topic: machine-learning
- Number of repositories: 50
- No date filtering

```
# Data Collection with detailed progress reporting
print(f"\n🔍 Searching GitHub for '{topic}' repositories...")
```

```
print(f"\n🔍 Searching GitHub for '{topic}' repositories...")
query = f"topic:{topic}+stars:>10"
if days_back > 0:
    date_filter = (datetime.now() - timedelta(days=days_back)).strftime("%Y-%m-%d")
    query += f"+pushed:>={date_filter}"

print(f"✓ Search query: {query}")
repos = []
page = 1
start_time = time.time()

try:
    while len(repos) < num_repos:
        print(f"📊 Fetching page {page}... ({len(repos)}/{num_repos} repos collected)")
        url = f"https://api.github.com/search/repositories?q={query}&sort=updated&order=d

        response = requests.get(url, headers=headers)
        if response.status_code == 200:
            data = response.json()
            new_repos = data['items']

            if not new_repos:
                print("ℹ️ No more repositories found")
                break

            print(f"✓ Found {len(new_repos)} repositories on page {page}")
            repos += [repo for repo in new_repos if repo['full_name'] not in [r['full_nam
            print(f"✓ Total unique repositories: {len(repos)}")

            # Check rate limits
            if 'X-RateLimit-Remaining' in response.headers:
                remaining = int(response.headers['X-RateLimit-Remaining'])
                if remaining < 10:
                    print(f"⚠️ API rate limit approaching: {remaining} requests remaining

            page += 1
            time.sleep(1.5)
        elif response.status_code == 403:
            reset_time = int(response.headers.get('X-RateLimit-Reset', 0))
            wait_time = max(0, reset_time - time.time())
            print(f"⚠️ Rate limit exceeded. Reset in {wait_time:.0f} seconds.")
            if wait_time > 180:
                print("⚠️ Rate limit wait too long, proceeding with collected data")
                break
            elif wait_time > 0 and wait_time < 180:
                print(f"⏳ Waiting {wait_time:.0f} seconds for rate limit reset...")
                time.sleep(wait_time + 1)
        else:
            print(f"❌ Error: {response.status_code} - {response.json().get('message', 'U
            break
except Exception as e:
    print(f"❌ Exception during repository collection: {str(e)}")

repos = repos[:num_repos]
if len(repos) < num_repos:
    print(f"⚠️ Could only collect {len(repos)} repositories (requested {num_repos})")
else:
    print(f"✅ Successfully collected {len(repos)} repositories")
```

```
elapsed = time.time() - start_time
print(f"⏱️ Repository collection completed in {elapsed:.1f} seconds")
```

```
🔍 Searching GitHub for 'machine-learning' repositories...
✓ Search query: topic:machine-learning+stars:>10
📊 Fetching page 1... (0/50 repos collected)
✓ Found 100 repositories on page 1
✓ Total unique repositories: 100
✅ Successfully collected 50 repositories
⏱️ Repository collection completed in 4.2 seconds
```

```
# Display repositories
print("\n📊 Repository Statistics:")
repo_df = pd.DataFrame([{
    'Repository': repo['full_name'],
    'Stars': repo['stargazers_count'],
    'Forks': repo['forks_count'],
    'Language': repo['language'] or 'Not specified',
    'Updated': repo['pushed_at'][:10],
    'Description': (repo['description'] or '')[:50] + ('...' if repo['description'] and le
} for repo in repos])

repo_df['Star Rating'] = repo_df['Stars'].apply(lambda x: '★' * min(5, max(1, int(np.log10
repo_df
```

```
📊 Repository Statistics:
```

	Repository	Stars	Forks	Language	Updated	Description	Star Rating
0	Footballotter/Trading-GPT	15	2	Not specified	2025-05-04	TradeGPT is an intelligent trading bot built with ...	★
1	aai-institute/pyDVL	124	7	Python	2025-05-04	pyDVL is a library of stable implementations of al...	★★
2	emoss08/Trenova	30	9	Go	2025-05-04	An AI-driven asset based Transportation Management...	★
3	amitshekhariitbhu/machine-learning-interview-questions	11	1	Markdown	2025-05-04	Your Cheat Sheet for Machine Learning Interview – ...	★
4	zjunlp/OceanGPT	47	5	Python	2025-05-04	[ACL 2024] OceanGPT: A Large Language Model for Oc...	★
5	DragonFive/cv_nlp_deeplearning	96	51	Jupyter Notebook	2025-05-04	用python做计算机视觉，人工智能，机器学习，深度学习等	★

6	LibrePhotos/librephotos	7287	330	Python	2025-05-04	A self-hosted open source photo management service...	★★★
7	Techtonique/ahead	23	6	R	2025-05-04	Univariate and multivariate time series forecastin...	★
8	myui/rtrec	16	0	Python	2025-05-04	An realtime recommendation system supporting onlin...	★
9	DanielMartensson/CControl	233	58	C	2025-05-04	Using advanced control and computer vision techniq...	★★
10	tensorzero/tensorzero	3896	255	Rust	2025-05-04	TensorZero creates a feedback loop for optimizing ...	★★★
11	ultralytics/ultralytics	40171	7785	Python	2025-05-04	Ultralytics YOLO11 🚀	★★★★
12	hosseinmoein/DataFrame	2681	331	C++	2025-05-04	C++ DataFrame for statistical, Financial, and ML a...	★★★
13	marimo-team/learn	156	23	Python	2025-05-04	📚 A curated collection of marimo notebooks for edu...	★★
14	superiorlu/AITreasureBox	582	80	Ruby	2025-05-04	🤖 Collect practical AI repos, tools, websites, pap...	★★
15	EdAbati/outlines-haystack	11	0	Python	2025-05-04	Use `outlines` generators with Haystack.	★
16	cmunch1/nba-prediction	252	41	Jupyter Notebook	2025-05-04	A project to deploy an online app that predicts th...	★★
17	dusty-nv/jetson-containers	3015	592	Jupyter Notebook	2025-05-04	Machine Learning Containers for NVIDIA Jetson and ...	★★★
18	photoprism/photoprism	37206	2061	Go	2025-05-04	AI-Powered Photos App for the Decentralized Web 🌈💎...	★★★★
19	pytorch/pytorch	89632	24045	Python	2025-05-04	Tensors and Dynamic neural networks in Python with...	★★★★
20	faizan1234567/Brain-Tumors-Segmentation	80	4	Python	2025-05-04	Multimodal Brain mpMRI segmentation on BraTS 2023 ...	★

36	open-spaced-repetition/srs-benchmark	95	13	Jupyter Notebook	2025-05-04	A benchmark for spaced repetition schedulers/algor... ★
37	timothepearce/synda	41	10	Python	2025-05-04	A CLI for generating synthetic data ★
38	paperless-ngx/paperless-ngx	27121	1584	Python	2025-05-04	A community-supported supercharged ★★★★

```
# Statistics summary
print(f"📈 Repository Stats Summary:")
print(f"- Most popular: {repo_df.iloc[repo_df['Stars'].idxmax()]['Repository']} ({repo_df
print(f"- Average stars: {repo_df['Stars'].mean():.1f}")
print(f"- Languages: {', '.join(repo_df['Language'].value_counts().index[:5])}")
```

```
📈 Repository Stats Summary:
- Most popular: pytorch/pytorch (89632 stars)
- Average stars: 8054.1
- Languages: Python, Jupyter Notebook, Not specified, C++, C
```

Contributors Collection

```
# @title Contributors Collection
contributors = defaultdict(lambda: {'repos': set(), 'contributions': 0})
repo_contributors = {}

print("\n👥 Collecting contributor data...")
contributor_start = time.time()
total_repos = len(repos)

for idx, repo in enumerate(repos):
    repo_name = repo['full_name']
    print(f"[{idx+1}/{total_repos}] Processing {repo_name}...")

    contributors_url = f"https://api.github.com/repos/{repo_name}/contributors?per_page=10
    page_count = 0
    contributor_count = 0

    try:
        while contributors_url:
            page_count += 1
            print(f"  Fetching contributors page {page_count}...")

            response = requests.get(contributors_url, headers=headers)
            if response.status_code != 200:
                print(f"  ❌ Error {response.status_code}: {response.json().get('message',
                break

            data = response.json()
            if not data:
                print("  ℹ️ No contributors found on this page")
                break

            new_contributors = len(data)
            contributor_count += new_contributors
```

```
            # print(f"  ✓ Found {new_contributors} contributors on page {page_count}")

            for contrib in data:
                login = contrib['login']
                contributions = contrib['contributions']
                contributors[login]['repos'].add(repo_name)
                contributors[login]['contributions'] += contributions

            if repo_name not in repo_contributors:
                repo_contributors[repo_name] = []
            repo_contributors[repo_name].extend([c['login'] for c in data])

            contributors_url = response.links.get('next', {}).get('url')

            if contributors_url:
                if 'X-RateLimit-Remaining' in response.headers:
                    remaining = int(response.headers.get('X-RateLimit-Remaining', 1))
                    if remaining < 5:
                        reset_time = int(response.headers.get('X-RateLimit-Reset', 0))
                        wait_time = max(0, reset_time - time.time())
                        print(f"  ⚠️ Rate limit low ({remaining} left). Reset in {wait_tim
                        if wait_time < 60:
                            print(f"  ⏳ Waiting {wait_time:.0f} seconds...")
                            time.sleep(wait_time + 1)

                time.sleep(0.8)
    except Exception as e:
        print(f"  ❌ Exception processing {repo_name}: {str(e)}")

    print(f"  ✅ Found {contributor_count} contributors for {repo_name}")

contributor_elapsed = time.time() - contributor_start
print(f"\n🧑‍💻 Found {len(contributors)} unique contributors across all repositories")
print(f"⏱️ Contributor collection completed in {contributor_elapsed:.1f} seconds")
```

Show hidden output

```
# Contributor stats
contributions_list = [data['contributions'] for data in contributors.values()]
avg_contributions = np.mean(contributions_list)
max_contributions = np.max(contributions_list)
print(f"📊 Contributor Stats:")
print(f"- Average contributions per user: {avg_contributions:.1f}")
print(f"- Maximum contributions by a user: {max_contributions}")
print(f"- Users with 10+ contributions: {len([c for c in contributions_list if c >= 10])}
```

```
📊 Contributor Stats:
- Average contributions per user: 116.9
- Maximum contributions by a user: 161822
- Users with 10+ contributions: 1344
```

Network Construction

```
# Network Construction
```

```
# Network construction
print("\n🧩 Building collaboration network...")
network_start = time.time()

G = nx.Graph()
print("📋 Adding contributor nodes...")

for i, (user, data) in enumerate(contributors.items()):
    if i % 100 == 0 and i > 0:
        print(f"  ✓ Added {i}/{len(contributors)} nodes...")

    G.add_node(user,
               repos=', '.join(sorted(data['repos'])),
               num_repos=len(data['repos']),
               contributions=int(data['contributions']),
               degree_cent=0.0,
               betweenness=0.0,
               community=-1)

print(f"✓ Added {len(G.nodes)} nodes to the network")
```

Show hidden output

```
# Add edges
print("\n📋 Adding collaboration edges...")
edge_counts = defaultdict(int)
edge_count = 0
repo_count = len(repo_contributors)

for idx, (repo, contribs) in enumerate(repo_contributors.items()):
    if idx % 5 == 0:
        print(f"  Processing repository {idx+1}/{repo_count}...")

    for i in range(len(contribs)):
        for j in range(i+1, len(contribs)):
            u1, u2 = contribs[i], contribs[j]
            edge_key = tuple(sorted((u1, u2)))
            edge_counts[edge_key] += 1
            edge_count += 1

print(f"  ✓ Found {edge_count} potential collaborations")
print(f"  ✓ Found {len(edge_counts)} unique collaboration pairs")

for idx, ((u1, u2), weight) in enumerate(edge_counts.items()):
    if idx % 1000 == 0 and idx > 0:
        print(f"  Added {idx}/{len(edge_counts)} edges...")

    G.add_edge(u1, u2, weight=int(weight))

network_elapsed = time.time() - network_start
print(f"\n🔗 Network construction complete in {network_elapsed:.1f} seconds")
print(f"📊 Network stats: {G.number_of_nodes()} nodes, {G.number_of_edges()} edges")
```

Show hidden output

Connectivity analysis

```
# @title Connectivity analysis
components = list(nx.connected_components(G))
print(f"📋 Network contains {len(components)} connected components")
print(f"  - Largest component has {len(max(components, key=len))} nodes ({len(max(compone
```

```
📋 Network contains 19 connected components
  - Largest component has 3725 nodes (90.5% of network)
```

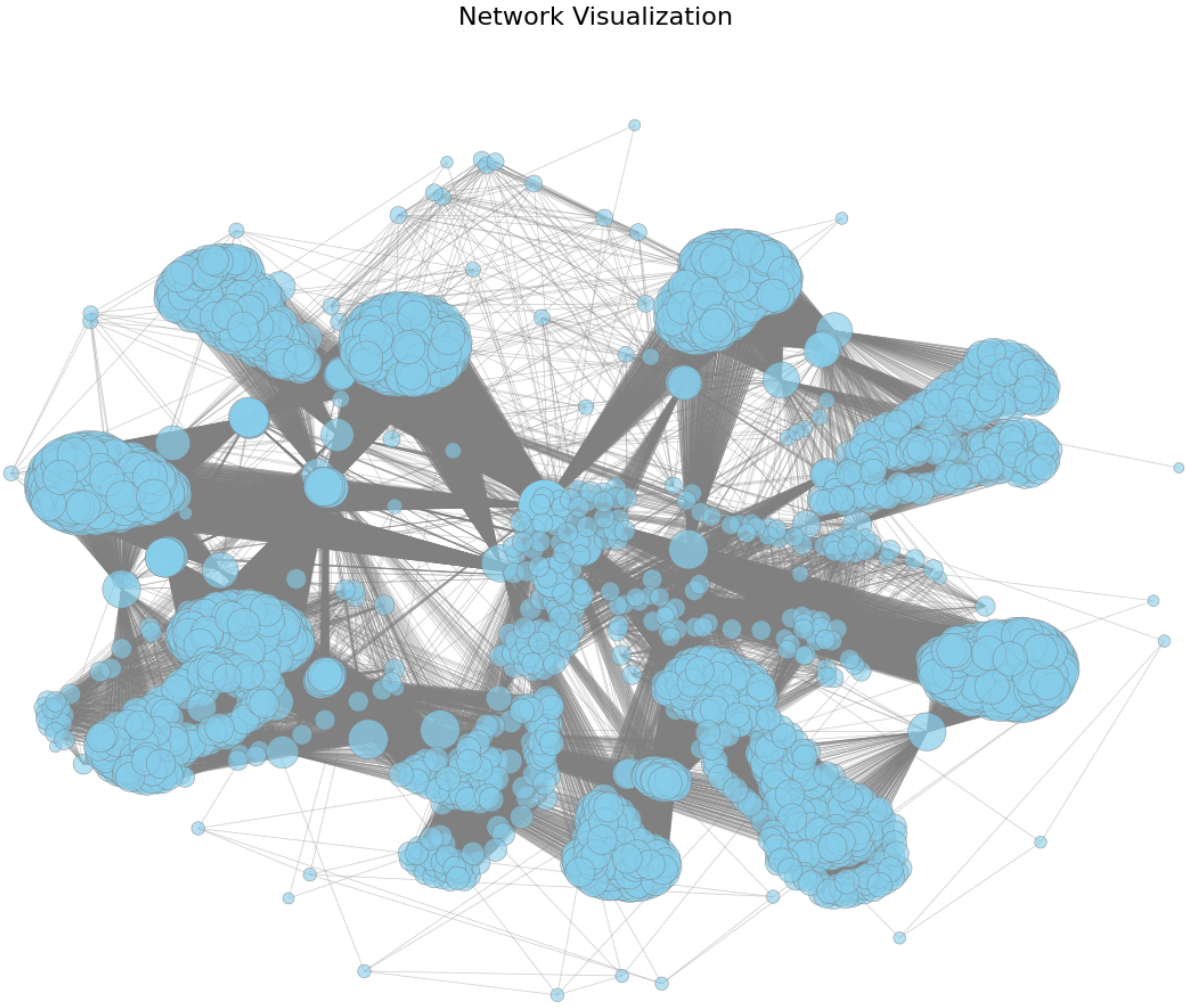
Network Analysis

Visualizing the Network

```
# Simple Visualization
def create_simple_vis():
  print("\n🟢 Generating simple network visualization...")
  viz_start = time.time()

  # Use largest connected component
  components = list(nx.connected_components(G))
  main_component = G.subgraph(max(components, key=len)).copy()
  print(f"📊 Visualizing main component with {main_component.number_of_nodes()} nodes and

  # Calculate layout
  print("📐 Computing layout...")
  pos = nx.spring_layout(main_component, k=0.3, seed=42, iterations=50)

  # Degree-based sizing
  degrees = dict(nx.degree(main_component))
  max_degree = max(degrees.values()) if degrees else 1
  node_size = [np.sqrt(d * 500) + 20 for d in degrees.values()]
  node_alpha = [0.6 + 0.4*(d/max_degree) for d in degrees.values()]

  # Plot
  plt.figure(figsize=(16, 12))
  ax = plt.gca()

  # Draw edges
  nx.draw_networkx_edges(
      main_component,
      pos,
      width=0.5,
      alpha=0.4,
      edge_color='gray'
  )

  # Draw nodes (uniform color)
  nx.draw_networkx_nodes(
      main_component,
      pos,
      node_size=node_size,
      alpha=node_alpha,
```

```
      node_color='skyblue',
      edgecolors='gray',
      linewidths=0.3
  )

  plt.title("Network Visualization", fontsize=16)
  plt.axis('off')
  plt.show()

  print(f"✅ Visualization complete in {time.time() - viz_start:.2f} seconds.")
create_simple_vis();
```

```
🟢 Generating simple network visualization...
📊 Visualizing main component with 3725 nodes and 511134 edges
📐 Computing layout...
```




```
✅ Visualization complete in 68.25 seconds.
```

Centrality Metrics

```
print("\n📊 Performing network analysis...")
analysis_start = time.time()
```

```
📊 Performing network analysis...
```

```
# Degree centrality
print("📈 Calculating degree centrality...")
degree_start = time.time()
degree_cent = nx.degree_centrality(G)
degree_elapsed = time.time() - degree_start
print(f"  ✓ Completed in {degree_elapsed:.2f} seconds")
nx.set_node_attributes(G, degree_cent, 'degree_cent')

print()
print("Degree Centrality ")
degree_cent_df = pd.DataFrame(list(degree_cent.items()), columns=['User', 'Degree Central
degree_cent_df.sort_values(by='Degree Centrality', ascending=False, inplace=True)
degree_cent_df
```

```
📈 Calculating degree centrality...
  ✓ Completed in 0.00 seconds

Degree Centrality
```

	User	Degree Centrality
10	dependabot[bot]	0.555744
3194	holdenk	0.198931
2910	JDarDagran	0.198931
1412	nishi-t	0.188487
1324	comaniac	0.188487
...	...	...
1697	BobMcDear	0.000000
82	myui	0.000000
1698	AshishKumar4	0.000000
961	faizan1234567	0.000000

0	Footballotter	0.000000

4118 rows × 2 columns

```
# Betweenness centrality
print("📈 Calculating betweenness centrality...")
betweenness_start = time.time()
k_sample = min(500, len(G.nodes))
print(f"  Using {k_sample} node samples for betweenness approximation")
betweenness = nx.betweenness_centrality(G, k=k_sample, seed=42)
betweenness_elapsed = time.time() - betweenness_start
print(f"  ✓ Completed in {betweenness_elapsed:.2f} seconds")
nx.set_node_attributes(G, betweenness, 'betweenness')

print()
print("Betweenness centrality ")
betweenness_df = pd.DataFrame(list(betweenness.items()), columns=['User', 'Betweenness Ce
betweenness_df.sort_values(by='Betweenness Centrality', ascending=False, inplace=True)
betweenness_df
```

```
📈 Calculating betweenness centrality...
  Using 500 node samples for betweenness approximation
  ✓ Completed in 43.49 seconds

Betweenness centrality
```

	User	Betweenness Centrality
10	dependabot[bot]	0.509468
1988	lamberta	0.048516
2022	mrkm4ntr	0.040002
1320	siju-samuel	0.034324
1529	tobegit3hub	0.034324
...	...	...
1403	mikepapadim	0.000000
1404	abhikran-quic	0.000000
1405	grant-arm	0.000000
1406	weberlo	0.000000
1391	ganler	0.000000

4118 rows × 2 columns

```
# PageRank
print("📈 Calculating PageRank...")
pagerank_start = time.time()
pagerank = nx.pagerank(G, alpha=0.85)
```

```
pagerank = nx.pagerank(G, alpha=0.85)
nx.set_node_attributes(G, pagerank, 'pagerank')
pagerank_elapsed = time.time() - pagerank_start
print(f"  ✓ Completed in {pagerank_elapsed:.2f} seconds")

print()
print("PageRank ")
pagerank_df = pd.DataFrame(list(pagerank.items()), columns=['User', 'PageRank'])
pagerank_df.sort_values(by='PageRank', ascending=False, inplace=True)
pagerank_df
```

```
📈 Calculating PageRank...
  ✓ Completed in 1.34 seconds
```

PageRank

	User	PageRank
10	dependabot[bot]	0.002321
7	github-actions[bot]	0.000722
728	vfdev-5	0.000638
837	thomasjpfan	0.000635
1798	omahs	0.000622
...	...	...
1698	AshishKumar4	0.000036
22	DragonFive	0.000036
82	myui	0.000036
961	faizan1234567	0.000036
0	Footballotter	0.000036

4118 rows × 2 columns

Community Detection

```
# Community detection using Louvain algorithm
print("🧩 Detecting communities with Louvain algorithm...")
comm_start = time.time()
communities = nx.community.louvain_communities(G, resolution=0.9, seed=42)
comm_elapsed = time.time() - comm_start
print(f"  ✓ Completed in {comm_elapsed:.2f} seconds")
print(f"  ✓ Detected {len(communities)} communities")
```

```
🧩 Detecting communities with Louvain algorithm...
  ✓ Completed in 4.49 seconds
  ✓ Detected 34 communities
```

```
# Map communities
```

```
# Map communities
community_map = {}
community_sizes = []
for i, comm in enumerate(communities):
    comm_size = len(comm)
    community_sizes.append(comm_size)
    print(f"  ✓ Community {i+1}: {comm_size} members")
    for user in comm:
        community_map[user] = int(i)

nx.set_node_attributes(G, community_map, 'community')
```

Show hidden output

```
# Critical nodes
print("🔍 Finding critical nodes and bridges...")
cut_nodes = list(nx.articulation_points(G))
bridges = list(nx.bridges(G))
print(f"  ✓ Found {len(cut_nodes)} articulation points")
print(f"  ✓ Found {len(bridges)} bridges")

analysis_elapsed = time.time() - analysis_start
print(f"\n✅ Network analysis completed in {analysis_elapsed:.1f} seconds")
```

```
🔍 Finding critical nodes and bridges...
  ✓ Found 9 articulation points
  ✓ Found 5 bridges

✅ Network analysis completed in 194.6 seconds
```

Enhanced Visualization

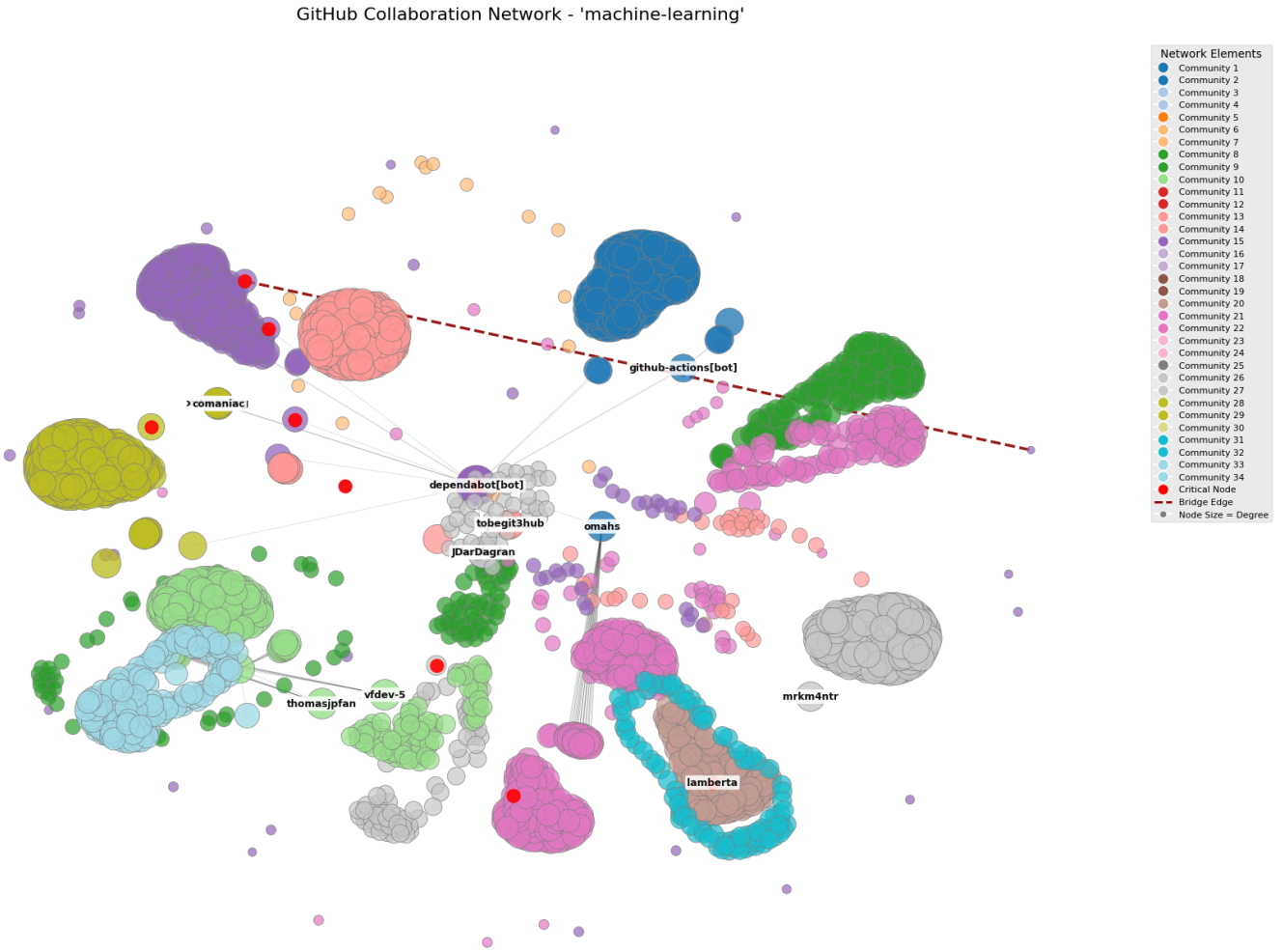
```
# Enhanced Visualization
print("\n🎨 Generating visualizations...")
viz_start = time.time()

plt.figure(figsize=(20, 16))
ax = plt.gca()

# Use largest connected component
components = list(nx.connected_components(G))
main_component = G.subgraph(max(components, key=len)).copy()
print(f"📊 Visualizing main component with {main_component.number_of_nodes()} nodes and {

# Calculate layout
print("📐 Computing network layout...")
layout_start = time.time()
pos = nx.spring_layout(main_component, k=0.3, seed=42, iterations=50)
layout_elapsed = time.time() - layout_start
print(f"  ✓ Layout computed in {layout_elapsed:.2f} seconds")

# Node styling
degrees = dict(nx.degree(main_component))
community_colors = [main_component.nodes[n]['community'] for n in main_component.nodes]
max_degree = max(degrees.values()) if degrees else 1
```

```
max_degree = max(degrees.values()) if degrees else 1
node_size = [np.sqrt(d * 750) + 20 for d in dict(main_component.degree()).values()]
node_alpha = [0.7 + 0.3*(d/max_degree) for d in degrees.values()]

# Draw nodes
print("🎨 Drawing nodes...")
cmap = plt.cm.get_cmap('tab20', len(communities))
nodes = nx.draw_networkx_nodes(
    main_component,
    pos,
    node_size=node_size,
    node_color=community_colors,
    cmap=cmap,
    alpha=node_alpha,
    edgecolors='gray',
    linewidths=0.5
)

# Draw edges
print("🎨 Drawing edges...")
edges = main_component.edges(data=True)
edge_widths = [np.log(data['weight'])*0.5 for _, _, data in edges]
edge_alphas = [min(0.2 + data['weight']*0.05, 0.6) for _, _, data in edges]

nx.draw_networkx_edges(
    main_component,
    pos,
    edgelist=edges,
    width=edge_widths,
    alpha=edge_alphas,
    edge_color='#444444'
)

# Highlight critical nodes
print("🔦 Highlighting critical elements...")
cut_nodes_main = [n for n in cut_nodes if n in main_component.nodes]
if cut_nodes_main:
    nx.draw_networkx_nodes(
        main_component,
        pos,
        nodelist=cut_nodes_main,
        node_size=150,
        node_color='red',
        alpha=0.9,
        label='Critical Nodes'
    )

# Highlight bridge edges
bridge_edges = [e for e in bridges if e[0] in main_component and e[1] in main_component]
if bridge_edges:
    nx.draw_networkx_edges(
        main_component,
        pos,
        edgelist=bridge_edges,
        width=2.5,
        alpha=0.9,
        edge_color='darkred',
        style='dashed',
        label='Bridge Edges'
```

```
        label='Bridge Edges'
    )

# Label important nodes
print("🏷️ Labeling key nodes...")
top_degree = sorted(degrees.items(), key=lambda x: x[1], reverse=True)[:7]
top_betweenness = sorted(betweenness.items(), key=lambda x: x[1], reverse=True)[:5]
top_pagerank = sorted(pagerank.items(), key=lambda x: x[1], reverse=True)[:5]
important_nodes = set([n[0] for n in top_degree + top_betweenness + top_pagerank])
labels = {n: n for n in important_nodes}

nx.draw_networkx_labels(
    main_component,
    pos,
    labels=labels,
    font_size=9,
    font_color='black',
    font_weight='bold',
    alpha=0.95,
    bbox=dict(
        facecolor='white',
        edgecolor='none',
        alpha=0.85,
        boxstyle='round,pad=0.2'
    )
)

# Create legend
print("📜 Creating legend...")
legend_elements = [
    plt.Line2D([0], [0],
               marker='o',
               color='w',
               label=f'Community {i+1}',
               markerfacecolor=cmap(i),
               markersize=10)
    for i in range(len(communities))
]

legend_elements += [
    plt.Line2D([0], [0],
               marker='o',
               color='w',
               label='Critical Node',
               markerfacecolor='red',
               markersize=10),
    plt.Line2D([0], [0],
               color='darkred',
               linewidth=2,
               label='Bridge Edge',
               linestyle='dashed'),
    plt.Line2D([0], [0],
               marker='o',
               color='w',
               label='Node Size = Degree',
               markerfacecolor='grey',
               markersize=np.sqrt(100)/1.5)
]
```

```
plt.legend(
    handles=legend_elements,
    loc='upper left',
    title="Network Elements",
    fontsize=8,
    title_fontsize=10,
    bbox_to_anchor=(1.01, 1),
    borderaxespad=0.
)

plt.title(
    f"GitHub Collaboration Network - '{topic}'\n", fontsize=16
)
plt.axis('off')
plt.show()
```

```
🎨 Generating visualizations...
📊 Visualizing main component with 3725 nodes and 511134 edges
📐 Computing network layout...
  ✓ Layout computed in 59.09 seconds
🎨 Drawing nodes...
🎨 Drawing edges...
<ipython-input-24-bbf3a2611702>:29: MatplotlibDeprecationWarning: The get_cmap functi
  cmap = plt.cm.get_cmap('tab20', len(communities))
/usr/local/lib/python3.11/dist-packages/networkx/drawing/nx_pylab.py:457: UserWarning
  node_collection = ax.scatter(
🔦 Highlighting critical elements...
🏷️ Labeling key nodes...
📜 Creating legend...
```



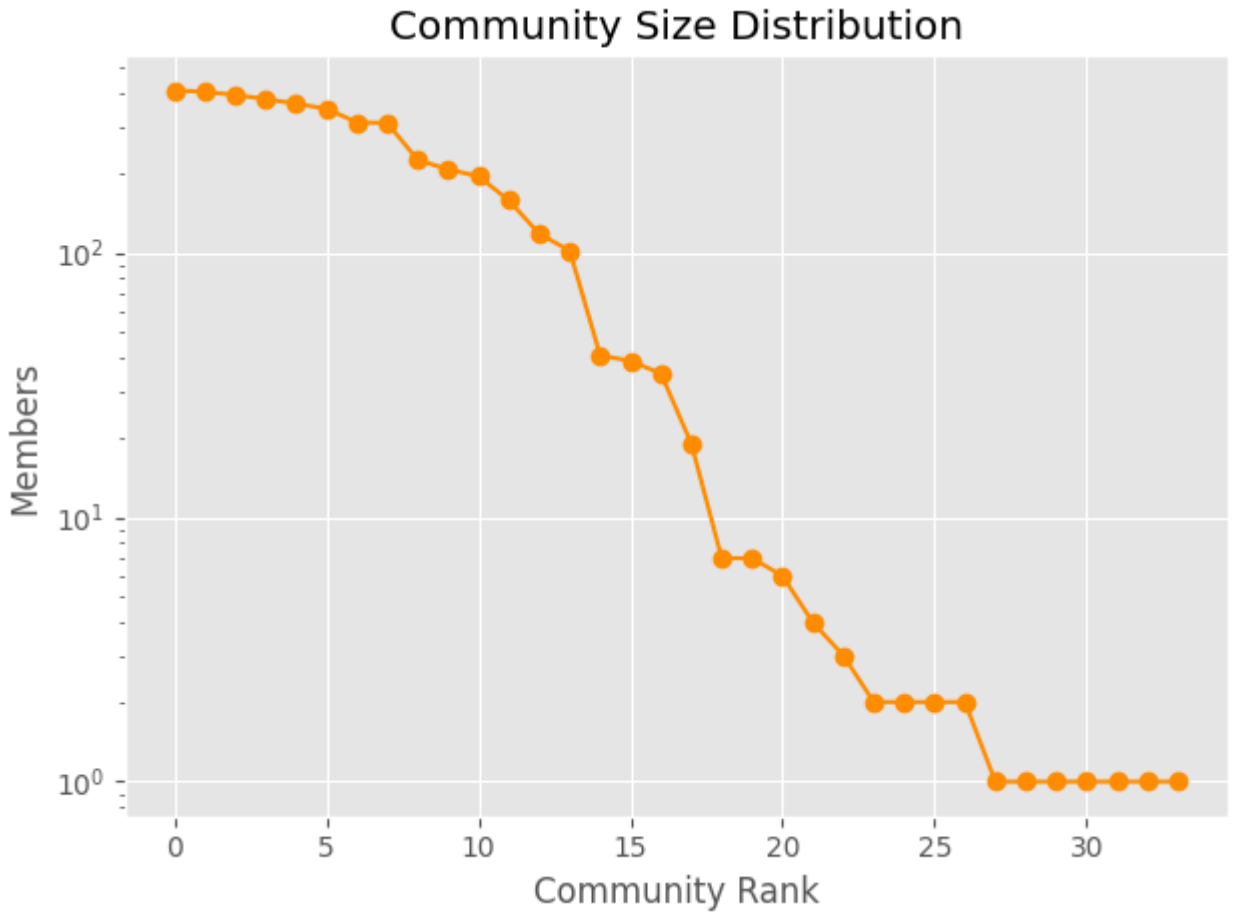
```
# Network Insights
stats_text = f"""Network Insights:
- {len(communities)} Communities Detected
- Largest Community: {max(community_sizes)} members
- {len(cut_nodes)} Critical Nodes
- {len(bridges)} Bridge Edges
- Average Degree: {np.mean(list(degrees.values())):.1f}
- Max Degree: {max(degrees.values())}"""
print(stats_text)
```

```
Network Insights:
- 34 Communities Detected
- Largest Community: 412 members
- 9 Critical Nodes
```

- 5 Bridge Edges
- Average Degree: 274.4
- Max Degree: 2288

Community Size Distribution

```
# Community Size Distribution
# plt.subplot(1, 3, 2)
community_sizes_sorted = sorted(community_sizes, reverse=True)
plt.plot(community_sizes_sorted, marker='o', linestyle='-', color='darkorange')
plt.title('Community Size Distribution')
plt.xlabel('Community Rank')
plt.ylabel('Members')
plt.yscale('log')
plt.tight_layout()
plt.show()
```

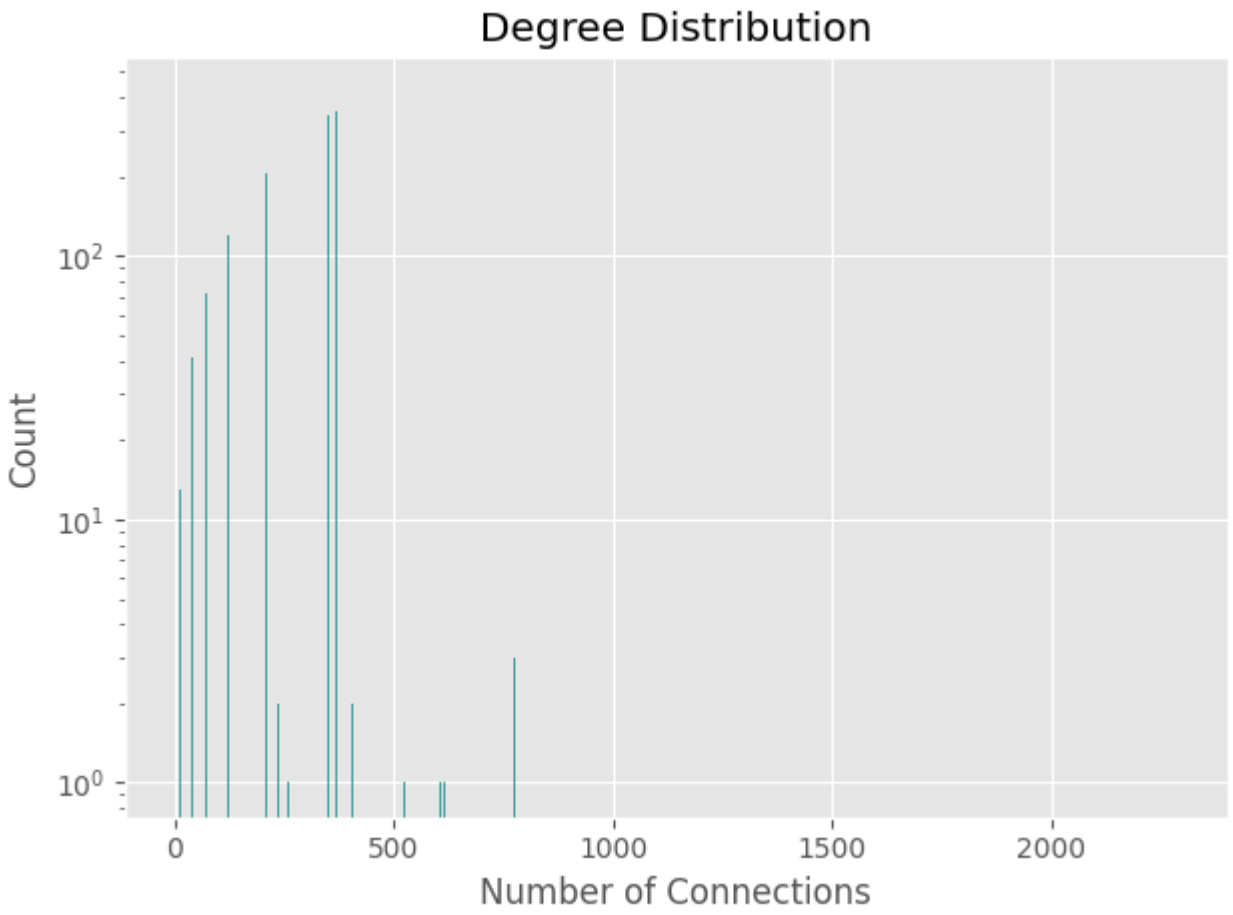


Community Size Distribution

- Each dot represents a community detected in the network (using community detection algorithm Louvain).
- The plot shows a power-law decay, meaning a few communities are very large, while most are small.
- This reflects a **hierarchical** or modular structure — common in real-world networks where people tend to form tight-knit groups.

Degree distribution

```
# Degree distribution
# plt.figure(figsize=(18, 6))
# plt.subplot(1, 3, 1)
degrees = dict(nx.degree(main_component))
degree_hist = np.unique(list(degrees.values()), return_counts=True)
plt.bar(degree_hist[0], degree_hist[1], color='teal', alpha=0.7)
plt.title('Degree Distribution')
plt.xlabel('Number of Connections')
plt.ylabel('Count')
plt.yscale('log')
plt.tight_layout()
plt.show()
```



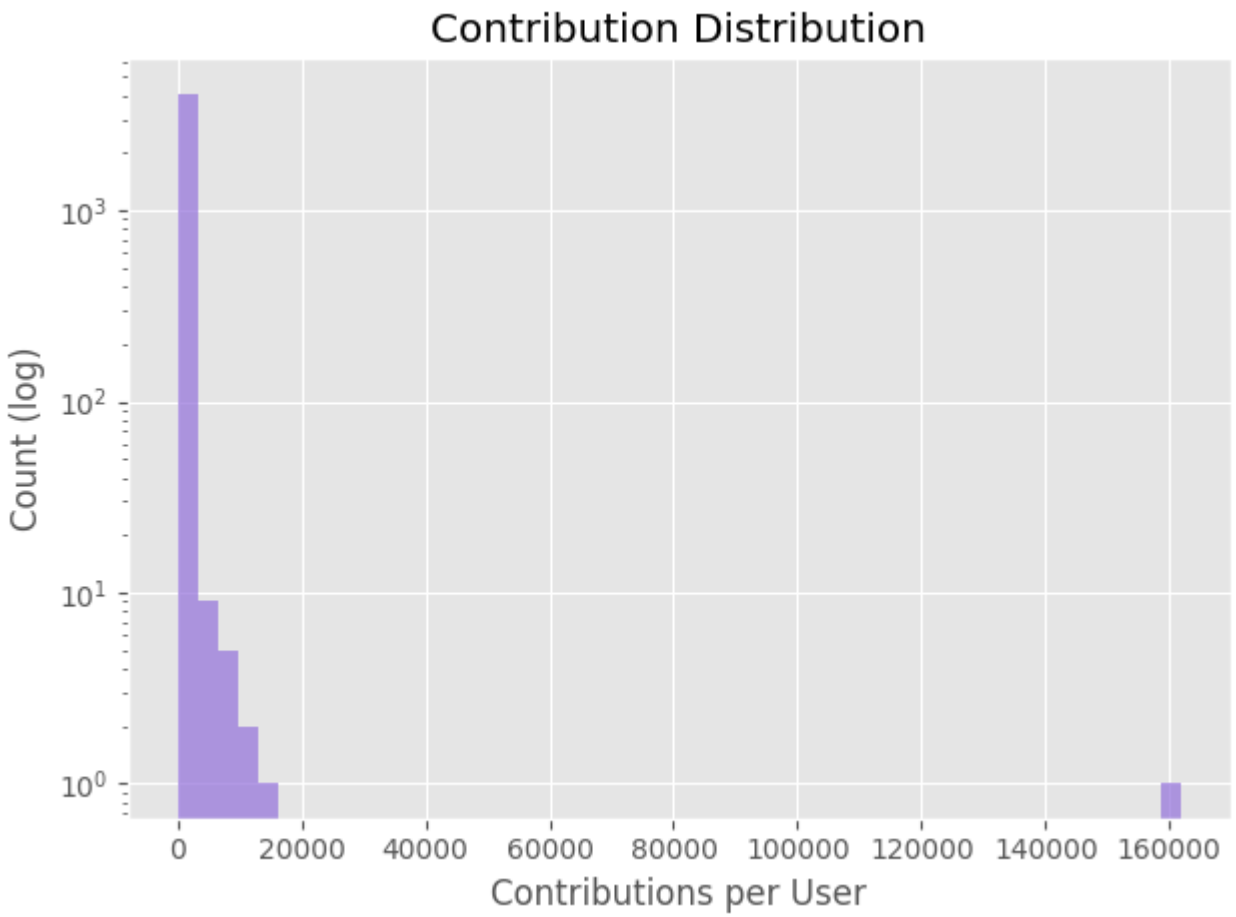
Degree Distribution

- This plot shows how connections are distributed across the network.
- It appears right-skewed, suggesting a heavy-tailed distribution: most nodes have few connections, while a few have very many.
- This is typical of **scale-free networks**, where a small number of hubs dominate the connectivity.

Contribution distribution

```
# Contribution distribution
# plt.subplot(1, 3, 3)
contributions = [c['contributions'] for c in contributors.values()]
plt.hist(contributions, bins=50, color='mediumpurple', alpha=0.7, log=True)
```

```
plt.title('Contribution Distribution')
plt.xlabel('Contributions per User')
plt.ylabel('Count (log)')
plt.tight_layout()
plt.show()
```



Contribution Distribution

- This histogram indicates that most users contribute rarely, but a tiny minority contribute heavily.
- The tail on the right shows some extreme outliers, with users making tens of thousands of contributions.
- Again, this resembles a Pareto principle (80-20 rule) where a small number of users drive most of the activity.

Small World Properties

```
# Small world properties
clustering_coefficient = nx.average_clustering(main_component)
shortest_path_length = nx.average_shortest_path_length(main_component)

print(f"Clustering coefficient: {clustering_coefficient}")
print(f"Average shortest path length: {shortest_path_length}")

# # Compare to a random graph
random_graph = nx.erdos_renyi_graph(main_component.number_of_nodes(), main_component.numb
random_clustering_coefficient = nx.average_clustering(random_graph)
random_shortest_path_length = nx.average_shortest_path_length(random_graph)

print(f"Clustering coefficient (random graph): {random_clustering_coefficient}")
```

```
print(f"Average shortest path length (random graph): {random_shortest_path_length}")

# Calculate characteristic path length (same as average shortest path length)
characteristic_path_length = shortest_path_length
print(f"Characteristic path length: {characteristic_path_length}")
```

```
Clustering coefficient: 0.9874526051519247
Average shortest path length: 2.4582644050202207
Clustering coefficient (random graph): 0.07356914158538568
Average shortest path length (random graph): 1.9263715857236572
Characteristic path length: 2.4582644050202207
```

```
# Calculate the small-world coefficient (sigma)
sigma = (clustering_coefficient/ random_clustering_coefficient)/ (shortest_path_length/ r
print(f"Small-world coefficient (Sigma): {sigma}")
```

```
Small-world coefficient (Sigma): 10.517972772660066
```

The network exhibits a very high small-world coefficient. Such a high σ suggests the network is highly modular and locally dense, with short global paths - a strong small-world structure.

Key findings

```
# Key findings
print("\n🔑 Key Network Insights:")
print(f"- Most connected user: {top_degree[0][0]} ({top_degree[0][1]} connections)")
print(f"- Most central user: {max(betweenness.items(), key=lambda x: x[1])[0]} (betweennes
print(f"- Largest community: {max(community_sizes)} members")
print(f"- Most active contributor: {max(contributors.items(), key=lambda x: x[1]['contribu
```

```
🔑 Key Network Insights:
- Most connected user: dependabot[bot] (2288 connections)
- Most central user: dependabot[bot] (betweenness: 0.509)
- Largest community: 412 members
- Most active contributor: Footballotter (161822 contributions)
```

Save Network

```
# Save network
nx.write_gexf(G, "github_collaboration_network.gexf")
```