Network Analysis for Telex: Improving Team Collaboration Through Smarter Communication

Introduction

What is Telex?

Telex is a team messaging and collaboration platform that helps employees and teams communicate better, manage projects smoothly, and share information quickly. The goal of Telex is to make workplace communication more organized, efficient, and smart by using AI to suggest messages, route conversations to the right people, and automate repetitive tasks. However, for Telex to work effectively, it needs real-world data to understand how people communicate in a professional setting. This is why we are using email network analysis to study how employees interact, so we can build Telex features that truly improve workplace messaging and workflow.

Why This Dataset?

To make Telex smarter, we need a dataset that mimics real workplace communication. The dataset we chose comes from a European research institution, where thousands of employees send and receive emails daily. This dataset provides two key files:

- email-Eu-core.txt – This file shows who sent emails to whom, helping us build a communication network.
- email-Eu-core-department-labels.txt – This file contains department information for each employee, allowing us to see which teams communicate the most and how information flows between departments.

This dataset is perfect for our study because it represents real workplace communication patterns, similar to how employees interact on Telex. By analyzing this data, we can gain valuable insights into team messaging, helping us design features that improve collaboration and workflow automation in Telex.

Objective of the Analysis

The goal of this analysis is to understand how employees communicate and use these insights to improve Telex's messaging and workflow automation features. To achieve this, we focus on six key objectives:

1. **Identify Key Communicators** – Find employees who send and receive the most messages and act as central hubs in the communication network. These individuals ensure information spreads quickly across teams.

2. **Detect Hidden Influencers** – Some employees may not send many emails, but they control important communication pathways. We aim to find these silent influencers who help keep teams connected.

3. **Spot Communication Bottlenecks** – If a few employees handle too many emails, they can slow down responses and cause delays. We will identify these bottlenecks and suggest ways to distribute communication more efficiently.

4. **Analyze Cross-Team Collaboration** – Do employees mostly communicate within their departments, or do they frequently interact with other teams? Understanding this helps Telex recommend the right chat groups and improve interdepartmental communication.

5. **Discover Natural Team Structures** – Instead of relying on formal departments, we will use data to see how employees naturally form workgroups. This allows Telex to suggest AI-driven team chats based on real work relationships rather than organizational charts.

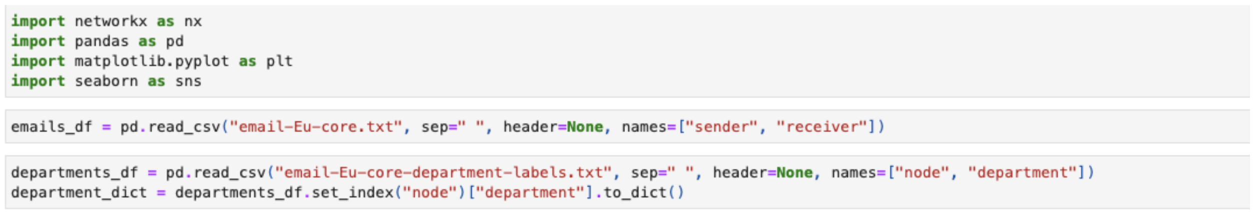
6. **Improve Message Routing & Workflow Automation** – By analyzing communication patterns, we can automate message distribution so that important information reaches the right people faster. This will help Telex create AI-powered message routing and smart notification systems that prevent employees from missing critical updates.

Data Preparation & Network Construction

To analyze workplace communication, we first load the dataset, build a network graph, and visualize the overall email flow. This helps us identify key communicators, bottlenecks, and collaboration patterns.

Loading the Data

The dataset includes two files: one tracking email interactions and another listing each employee's department. Using pandas, we load this data to examine who communicates with whom and how departments interact internally and externally. This prepares the information needed to study workplace messaging behavior.



```python
import networkx as nx
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```python
emails_df = pd.read_csv("email-Eu-core.txt", sep=" ", header=None, names=["sender", "receiver"])
```

```python
departments_df = pd.read_csv("email-Eu-core-department-labels.txt", sep=" ", header=None, names=["node", "department"])
department_dict = departments_df.set_index("node")["department"].to_dict()
```

Building the Network Graph

To better understand communication flow, we represent the email data as a directed graph using NetworkX, where:

- Nodes represent employees
- Edges represent emails sent from one employee to another

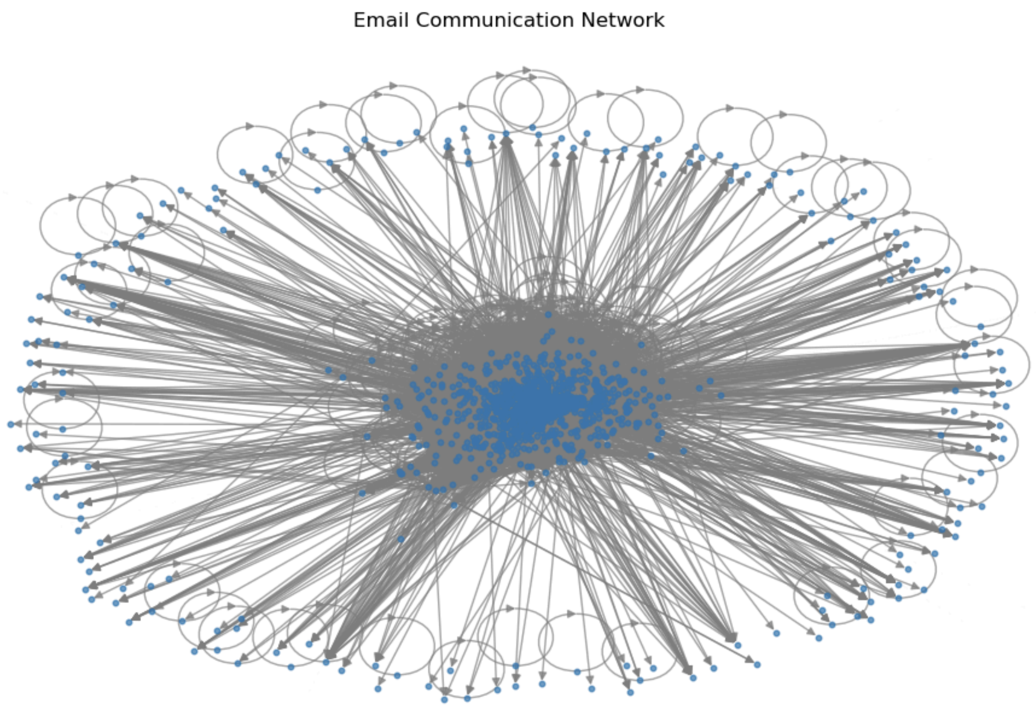
By calculating the number of connections (edges), we assess how collaborative or isolated the network is. A well-connected network suggests efficient communication, while gaps may indicate isolated teams. For Telex, this ensures optimized message flow and improved team collaboration.

```
G = nx.DiGraph()
G.add_edges_from(emails_df.values)
print(f"Graph has {G.number_of_nodes()} nodes and {G.number_of_edges()} edges.")
```

```
Graph has 1005 nodes and 25571 edges.
```

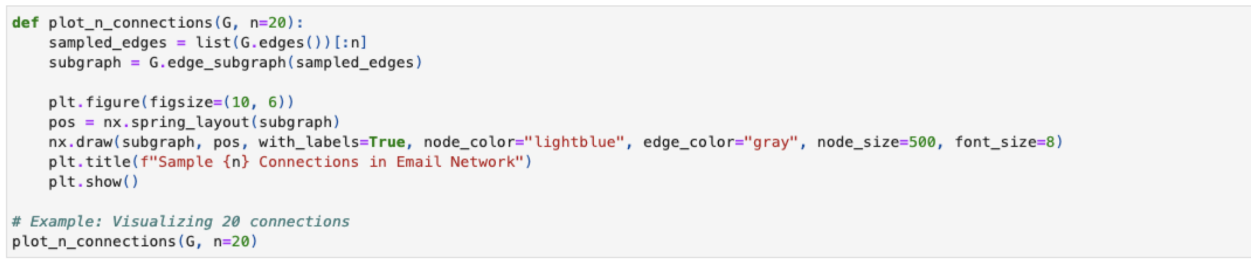
```
nx.set_node_attributes(G, department_dict, "department")
```

```
plt.figure(figsize=(10, 6))
nx.draw(G, node_size=10, edge_color="gray", alpha=0.7, with_labels=False)
plt.title("Email Communication Network")
plt.show()
```



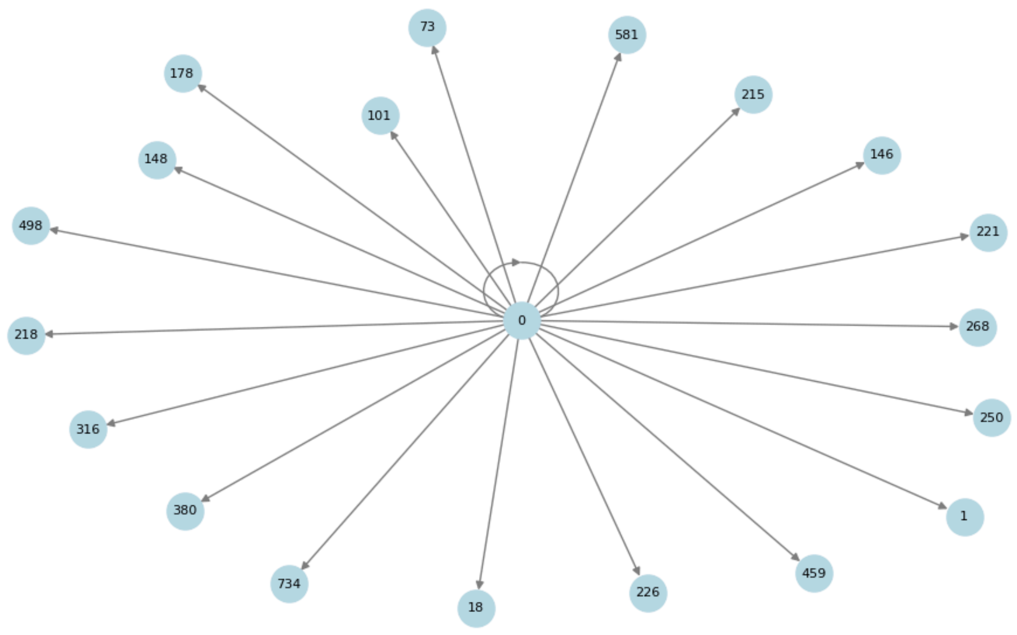
The Overall Email Network

To visualize the entire communication structure, we generate a network graph, where each node represents an employee, and each edge represents an email exchange. This helps us understand how connected the organization is, identifying strong communication links and potential isolated groups.



```python
def plot_n_connections(G, n=20):
    sampled_edges = list(G.edges())[:n]
    subgraph = G.edge_subgraph(sampled_edges)

    plt.figure(figsize=(10, 6))
    pos = nx.spring_layout(subgraph)
    nx.draw(subgraph, pos, with_labels=True, node_color="lightblue", edge_color="gray", node_size=500, font_size=8)
    plt.title(f"Sample {n} Connections in Email Network")
    plt.show()

# Example: Visualizing 20 connections
plot_n_connections(G, n=20)
```

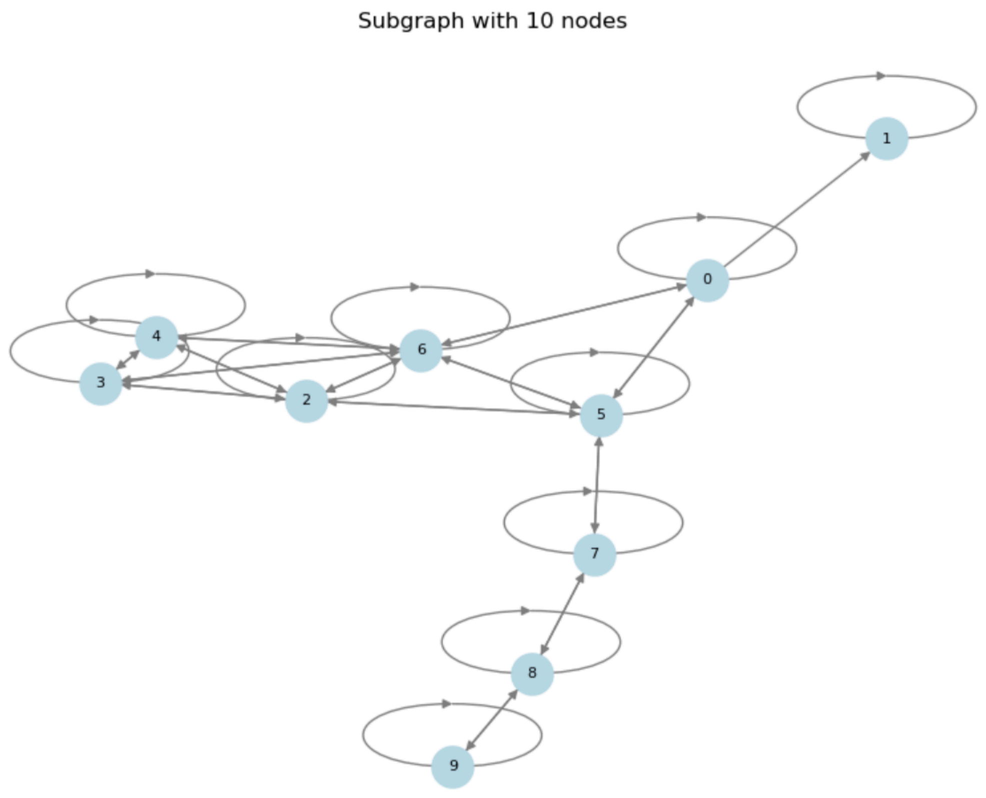
Sample 20 Connections in Email Network



Sampling Email Connections

The plot_n_connections() function provides a small-scale view of the email network by sampling n email exchanges (default: 20) and visualizing them as a subgraph. Each light blue node represents an employee, while gray lines show email exchanges. This helps spot-check communication flow, revealing frequent interactions, isolated employees, or team structures without the clutter of the full network. For Telex, this aids in debugging network structures, analyzing local interactions, and improving AI-powered message routing.

```python
def visualize_subgraph(n=10):
    sub_nodes = list(G.nodes)[:n]
    subgraph = G.subgraph(sub_nodes)

    plt.figure(figsize=(8, 6))
    nx.draw(subgraph, with_labels=True, node_color='lightblue', edge_color='gray', node_size=500, font_size=8)
    plt.title(f'Subgraph with {n} nodes')
    plt.show()

visualize_subgraph(10)  # Visualize first 10 connections
```



Key Research Questions & Findings

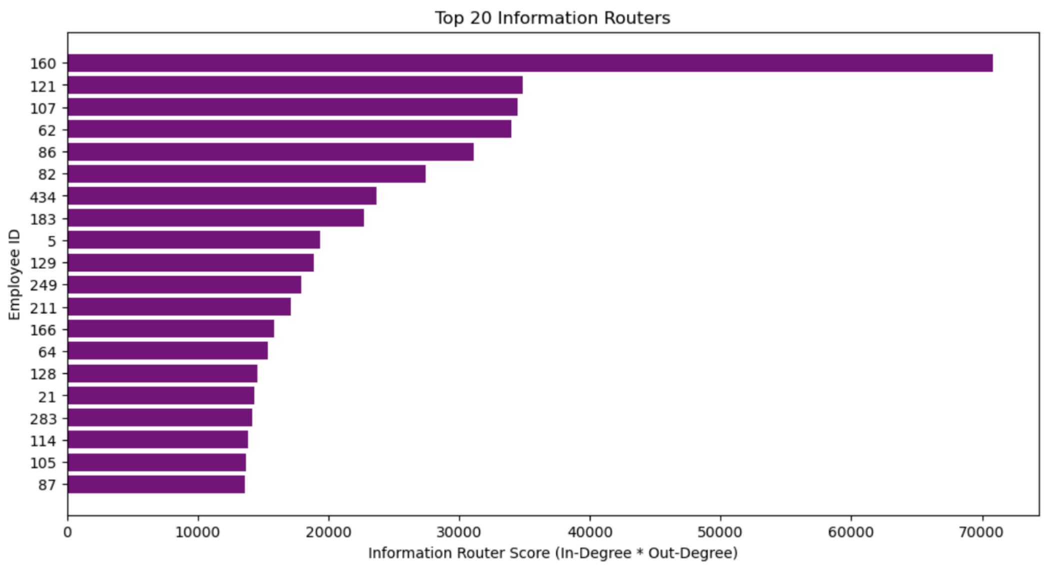
Each section will introduce the question, explain the visualization, interpret insights for Telex, and explain the code in layman's terms.

Who are the Information Routers in the Network?

Some employees act as message routers, receiving and forwarding emails to keep communication flowing between teams. If they stop forwarding messages, critical information may be delayed. The bar chart highlights the top 20 employees who play this role, with taller bars indicating key connectors. The code calculates how many emails each person receives (in-degree) and sends (out-degree), then multiplies these values to get an

"Information Router Score". The top 20 employees are displayed in a chart, showing who is most active in routing messages. For Telex, AI can auto-route messages to these individuals, preventing delays, and recommend spreading communication to avoid overload, making workflows faster and more efficient.

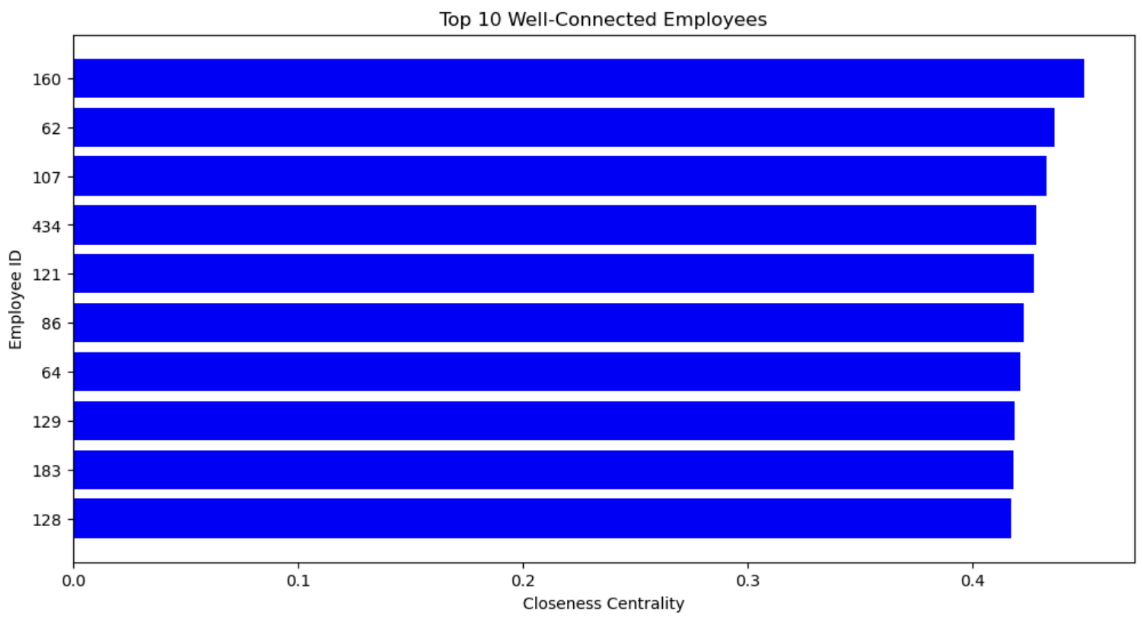
```python
# Compute in-degree (emails received) and out-degree (emails sent)
in_degree = dict(G.in_degree())
out_degree = dict(G.out_degree())

# Compute information router score = (in-degree * out-degree)
info_router_score = {node: in_degree[node] * out_degree[node] for node in G.nodes()}

# Get top 20 information routers
top_routers = sorted(info_router_score.items(), key=lambda x: x[1], reverse=True)[:20]

# Plot
plt.figure(figsize=(12, 6))
plt.barh([str(x[0]) for x in top_routers], [x[1] for x in top_routers], color="purple")
plt.xlabel("Information Router Score (In-Degree * Out-Degree)")
plt.ylabel("Employee ID")
plt.title("Top 20 Information Routers")
plt.gca().invert_yaxis()
plt.show()
```



Who Are the Most Well-Connected Employees?

Some employees are better positioned to spread information quickly because they can reach many others in just a few steps. These individuals have high closeness centrality, meaning they are well-connected across the organization and can deliver messages faster than others. The bar chart highlights the top 20 employees with the highest closeness scores, showing who plays a key role in fast communication flow. The code calculates closeness centrality for each employee, sorts them by highest score, and plots the top 20 in a bar chart. For Telex, these employees should be prioritized for urgent announcements, ensuring important messages reach the most people in the shortest time.

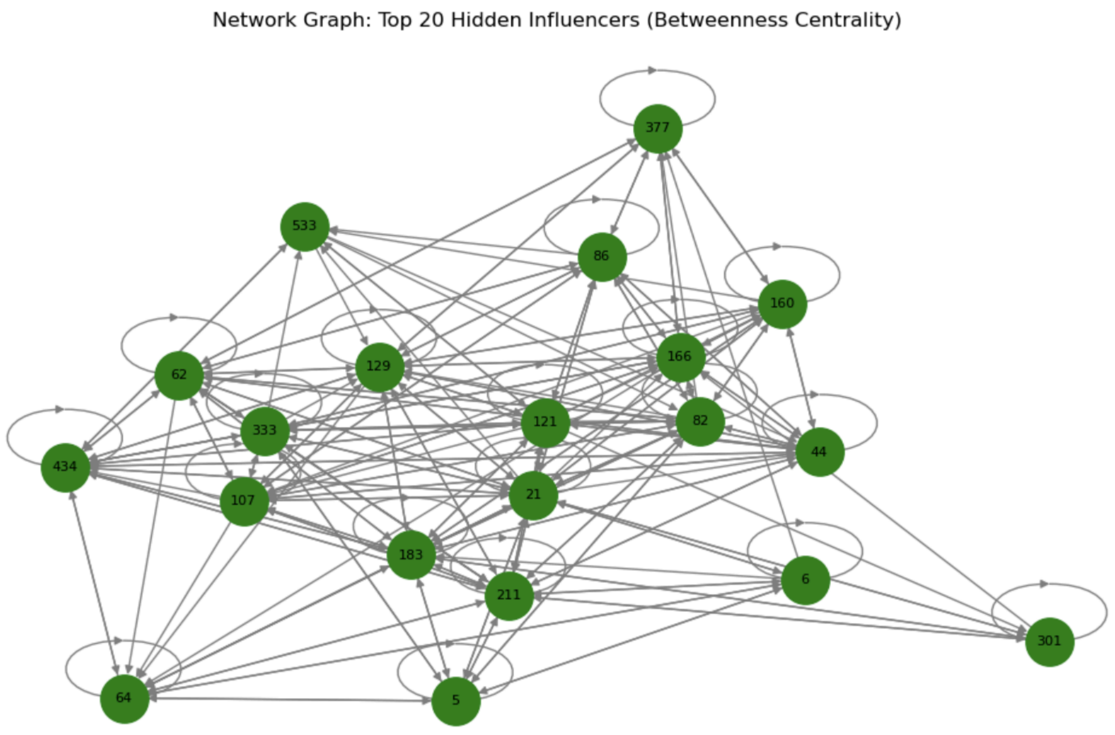
```
#Compute closeness centrality
closeness_centrality = nx.closeness_centrality(G)

# Get top 20 employees with the highest closeness centrality
top_closeness = sorted(closeness_centrality.items(), key=lambda x: x[1], reverse=True)[:10]

# Plot
plt.figure(figsize=(12, 6))
plt.barh([str(x[0]) for x in top_closeness], [x[1] for x in top_closeness], color="blue")
plt.xlabel("Closeness Centrality")
plt.ylabel("Employee ID")
plt.title("Top 10 Well-Connected Employees")
plt.gca().invert_yaxis()
plt.show()
```



Hidden Influencers: Employees Who Rarely Send Emails but Control Information Flow

Some employees don't send many emails, but they appear in many conversations, acting as silent bridges between teams. These hidden influencers ensure that information flows smoothly across the organization, even if they are not actively participating in discussions. The network graph highlights the top 20 hidden influencers and their direct connections, showing who plays a key role in linking different teams. The code calculates betweenness centrality, which identifies employees who frequently appear in the shortest paths between others. It then filters out those who don't send many emails (low out-degree) but still control key communication pathways. For Telex, AI should prioritize notifications for these employees, ensuring they stay updated on critical discussions, as they play a major role in keeping teams connected without actively driving conversations.

```
# Compute betweenness centrality
betweenness_centrality = nx.betweenness_centrality(G)

# Get top 20 hidden influencers based on betweenness centrality
top_betweenness = sorted(betweenness_centrality.items(), key=lambda x: x[1], reverse=True)[:20]

# Extract top influencer IDs
top_nodes = [x[0] for x in top_betweenness]

# Create a subgraph containing only the top 20 influencers and their connections
subG = G.subgraph(top_nodes)

# Visualize the network
plt.figure(figsize=(10, 6))
pos = nx.spring_layout(subG)  # Position nodes for clarity
nx.draw(subG, pos, with_labels=True, node_size=800, node_color="green", edge_color="gray", font_size=8)
plt.title("Network Graph: Top 20 Hidden Influencers (Betweenness Centrality)")
plt.show()
```

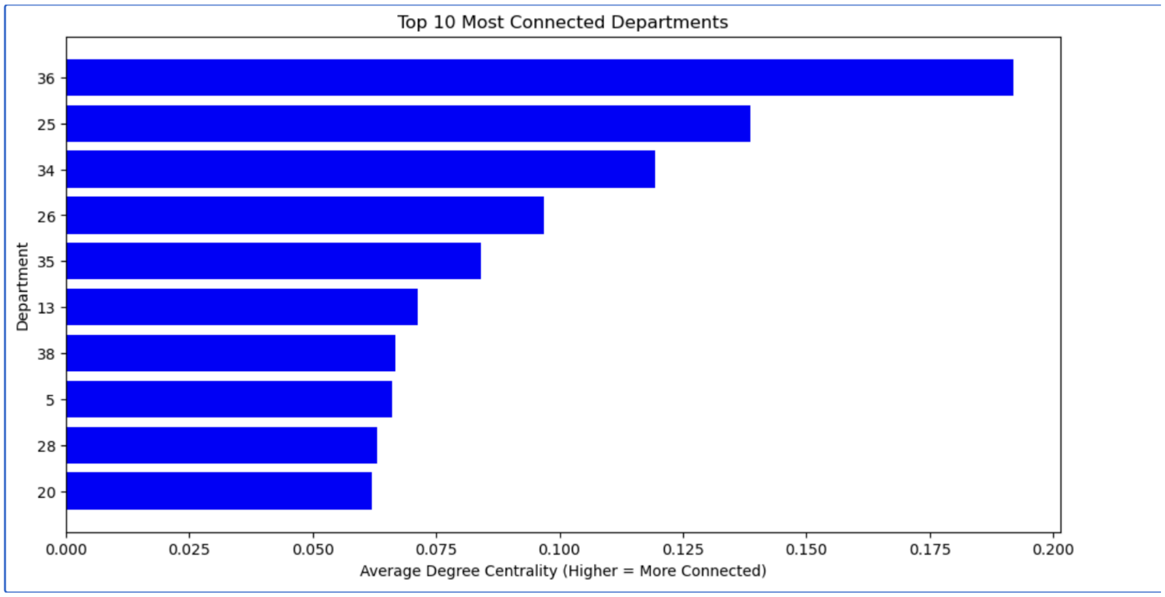

Network Graph: Top 20 Hidden Influencers (Betweenness Centrality)
377
533
86
160
62
129
166
333
121
82
44
434
107
21
183
6
211
301
64
5

Which Departments Have Centralized Communication?

In some departments, only a few employees handle most of the email communication, meaning important messages depend on a small group of people. If these key employees are unavailable, communication could slow down significantly. The bar chart ranks the most centralized departments, showing where a few individuals control the majority of information flow. The code calculates degree centrality (how many people each employee communicates with), and then finds the average centrality score for each department to see which ones rely heavily on a small number of employees. For Telex, this insight helps AI recommend better message distribution, ensuring that communication is more balanced across departments and not overly dependent on just a few individuals.

Analysis Approach: Degree Centrality. It directly measures how many connections a node has (how many people an employee communicates with).

```
In [16]: # Compute degree centrality for all employees
degree_centrality = nx.degree_centrality(G)

# Convert degree centrality to DataFrame
degree_centrality_df = pd.DataFrame(degree_centrality.items(), columns=["Employee", "Degree_Centrality"])

# Merge department information
degree_centrality_df = degree_centrality_df.merge(departments_df, left_on="Employee", right_on="node")

# Compute average degree centrality per department
dept_degree_centrality = degree_centrality_df.groupby("department")["Degree_Centrality"].mean().reset_index()

# Get the top 10 departments by average degree centrality
top_10_dept = dept_degree_centrality.nlargest(10, "Degree_Centrality")

# Plot
plt.figure(figsize=(12, 6))
plt.barh(top_10_dept["department"].astype(str), top_10_dept["Degree_Centrality"], color="blue")
plt.xlabel("Average Degree Centrality (Higher = More Connected)")
plt.ylabel("Department")
plt.title("Top 10 Most Connected Departments")
plt.gca().invert_yaxis()
plt.show()
```



How Do Teams Naturally Cluster?

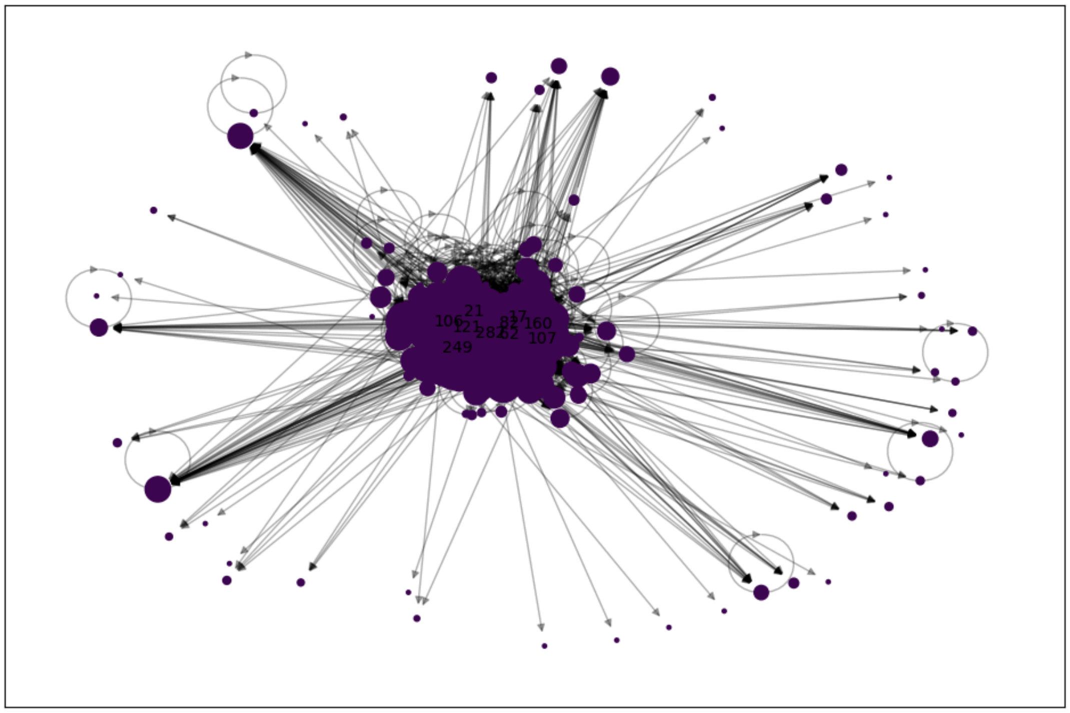
Employees don't always communicate strictly within their official departments—some teams naturally form based on real work interactions. These groups may consist of cross-functional teams, project-based collaborations, or informal work clusters. The network graph highlights the largest naturally formed communication cluster, showing how employees actually interact, regardless of department labels. The code uses the Louvain algorithm, which detects real communication patterns by grouping employees who frequently exchange emails. It then extracts the largest cluster and plots a network graph to visualize the strongest internal connections. For Telex, this insight allows AI to auto-suggest team chat groups based on real communication behavior, ensuring that employees are grouped in ways that reflect actual work relationships, rather than relying only on company structures.

```
import community as community_louvain

# Compute Louvain communities
partition = community_louvain.best_partition(G.to_undirected())

# Convert partition result into a DataFrame
cluster_df = pd.DataFrame(list(partition.items()), columns=["Employee", "Cluster"])

# Find the largest cluster
largest_cluster_id = cluster_df["Cluster"].value_counts().idxmax()  # Get the ID of the largest cluster
largest_cluster_nodes = cluster_df[cluster_df["Cluster"] == largest_cluster_id]["Employee"].tolist()

# Create a subgraph with only the **largest cluster**
subG = G.subgraph(largest_cluster_nodes)

# Compute network metrics
degree_centrality = nx.degree_centrality(subG)
betweenness_centrality = nx.betweenness_centrality(subG)
closeness_centrality = nx.closeness_centrality(subG)

# Create a DataFrame for the largest cluster
largest_cluster_df = pd.DataFrame({
    "Employee": list(subG.nodes()),
    "Degree": [subG.degree(n) for n in subG.nodes()],
    "Degree Centrality": [degree_centrality[n] for n in subG.nodes()],
    "Betweenness Centrality": [betweenness_centrality[n] for n in subG.nodes()],
    "Closeness Centrality": [closeness_centrality[n] for n in subG.nodes()]
})

# Sort by Degree Centrality (most connected first)
largest_cluster_df = largest_cluster_df.sort_values(by="Degree Centrality", ascending=False)

# Display the first 20 rows
print(largest_cluster_df.head(20))
```

```
     Employee  Degree  Degree Centrality  Betweenness Centrality  \
50        121     190           0.612903                0.056592   
41        107     186           0.600000                0.052639   
40        106     178           0.574194                0.042279   
12         62     170           0.548387                0.038961   
26         82     165           0.532258                0.034140   
7          21     155           0.500000                0.038072   
89        249     140           0.451613                0.016766   
99        282     139           0.448387                0.020411   
148       434     136           0.438710                0.020948   
4          17     134           0.432258                0.026253   
39        105     132           0.425806                0.013940   
54        142     128           0.412903                0.021363   
30         87     126           0.406452                0.019616   
21         74     125           0.403226                0.024927   
75        212     125           0.403226                0.012707   
25         81     124           0.400000                0.022631   
27         83     113           0.364516                0.016437   
66        166     109           0.351613                0.020474   
136       404     108           0.348387                0.026370   
122       329     104           0.335484                0.017422   

     Closeness Centrality  
50               0.476326  
41               0.485585  
40               0.497402  
12               0.486636  
26               0.450553  
7                0.468387  
89               0.459766  
99               0.466444  
148              0.467413  
4                0.425806  
39               0.463558  
54               0.458828  
30               0.455113  
21               0.436555  
75               0.453278  
25               0.453278  
27               0.438257  
66               0.453278  
136              0.424200  
122              0.439972  
```

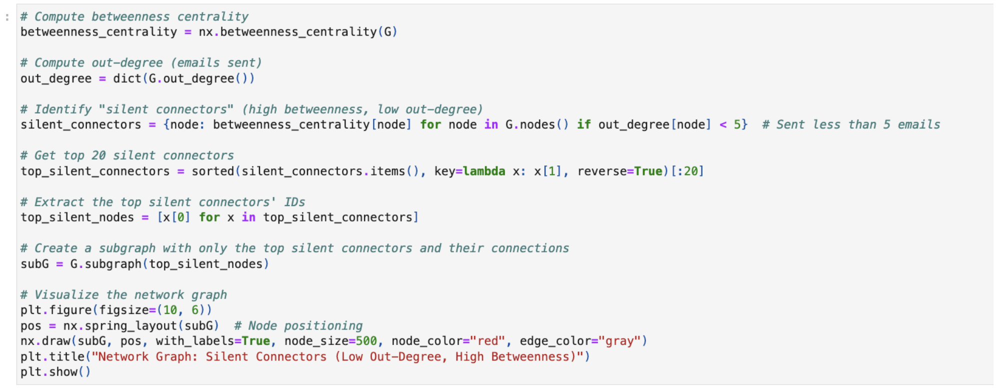
In [19]:
```python
import community as community_louvain

# Compute Louvain communities
partition = community_louvain.best_partition(G.to_undirected())

# Convert partition result into a DataFrame
cluster_df = pd.DataFrame(list(partition.items()), columns=["Employee", "Cluster"])

# Find the largest cluster
largest_cluster_id = cluster_df["Cluster"].value_counts().idxmax()  # Get the ID of the largest cluster
largest_cluster_nodes = cluster_df[cluster_df["Cluster"] == largest_cluster_id]["Employee"].tolist()

# Create a subgraph with only the **largest cluster**
subG = G.subgraph(largest_cluster_nodes)

# Compute degree centrality to scale node size
degree_centrality = nx.degree_centrality(subG)

# Draw the graph
plt.figure(figsize=(12, 8))
pos = nx.spring_layout(subG)  # Node positioning

# Draw nodes
nx.draw_networkx_nodes(subG, pos,
                       node_color=[partition[n] for n in subG.nodes()],
                       cmap=plt.get_cmap("viridis"),
                       node_size=[degree_centrality[n] * 2000 for n in subG.nodes()])

# Draw edges
nx.draw_networkx_edges(subG, pos, alpha=0.3)

# Draw labels for the most central nodes
important_nodes = sorted(degree_centrality, key=degree_centrality.get, reverse=True)[:10]  # Show top 10 labels
nx.draw_networkx_labels(subG, pos, labels={n: str(n) for n in important_nodes}, font_size=10, font_color="black")

# Show the graph
plt.title("Network Graph: Largest Communication Cluster")
plt.show()
```

Network Graph: Largest Communication Cluster



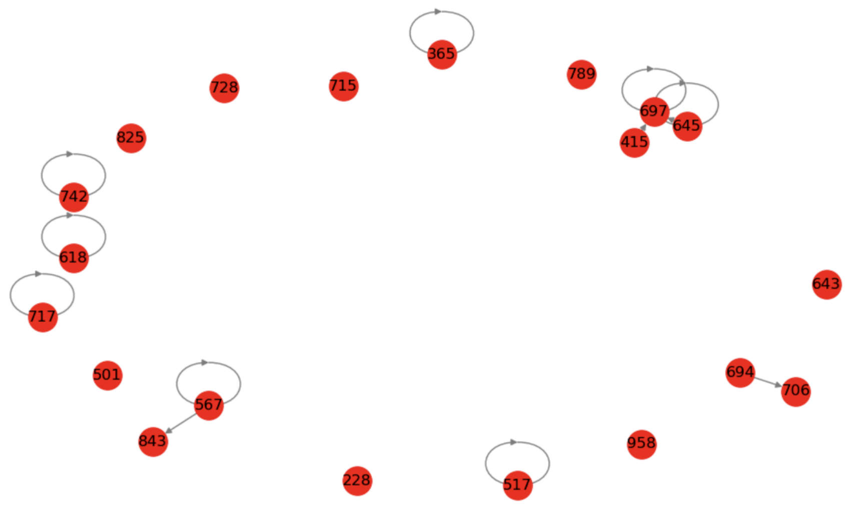
Which employees act as "silent connectors," bridging different teams but rarely initiating communication?

Silent Connectors and Hidden Influencers both impact communication but in different ways. Silent Connectors bridge separate teams but rarely send emails, while Hidden Influencers shape discussions without necessarily linking different groups. The code calculates betweenness centrality (how often an employee appears on shortest paths) and filters those with low out-degree to identify Silent Connectors, as they act as passive but essential links. Hidden Influencers also have high betweenness centrality, but they may send and receive emails more evenly, influencing conversations within teams rather than across them. For Telex, AI can ensure Silent Connectors stay informed by suggesting automatic updates, while Hidden Influencers receive prioritized notifications to keep workflows efficient.



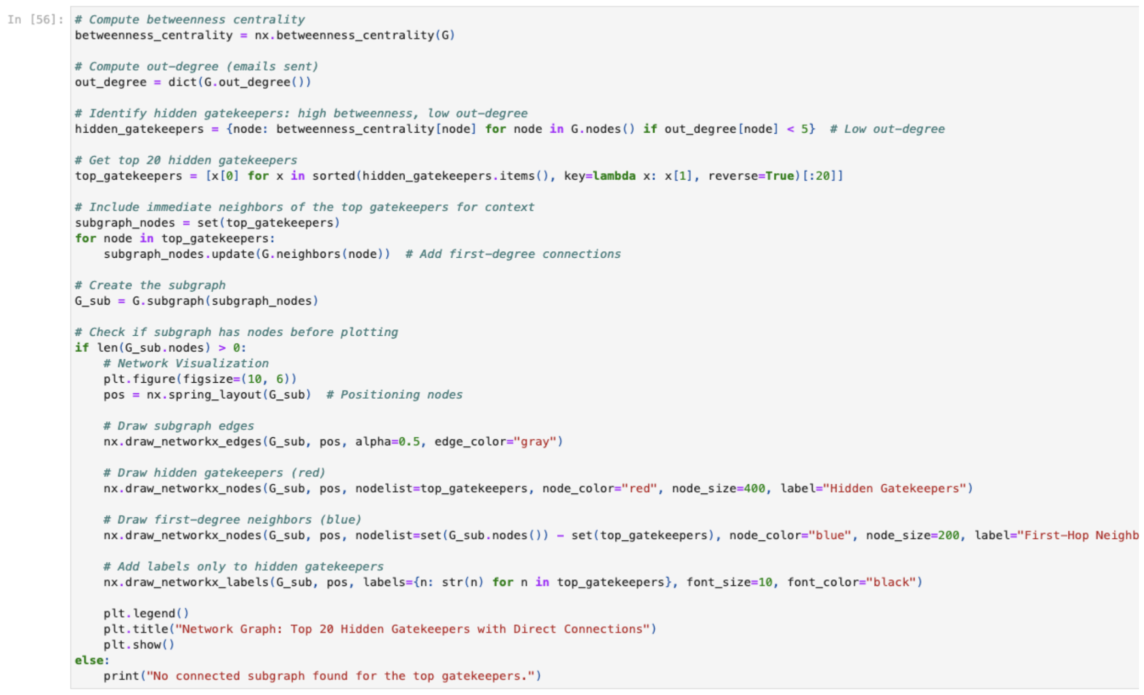
```python
# Compute betweenness centrality
betweenness_centrality = nx.betweenness_centrality(G)

# Compute out-degree (emails sent)
out_degree = dict(G.out_degree())

# Identify "silent connectors" (high betweenness, low out-degree)
silent_connectors = {node: betweenness_centrality[node] for node in G.nodes() if out_degree[node] < 5}  # Sent less than 5 emails

# Get top 20 silent connectors
top_silent_connectors = sorted(silent_connectors.items(), key=lambda x: x[1], reverse=True)[:20]

# Extract the top silent connectors' IDs
top_silent_nodes = [x[0] for x in top_silent_connectors]

# Create a subgraph with only the top silent connectors and their connections
subG = G.subgraph(top_silent_nodes)

# Visualize the network graph
plt.figure(figsize=(10, 6))
pos = nx.spring_layout(subG)  # Node positioning
nx.draw(subG, pos, with_labels=True, node_size=500, node_color="red", edge_color="gray")
plt.title("Network Graph: Silent Connectors (Low Out-Degree, High Betweenness)")
plt.show()
```

Network Graph: Silent Connectors (Low Out-Degree, High Betweenness)



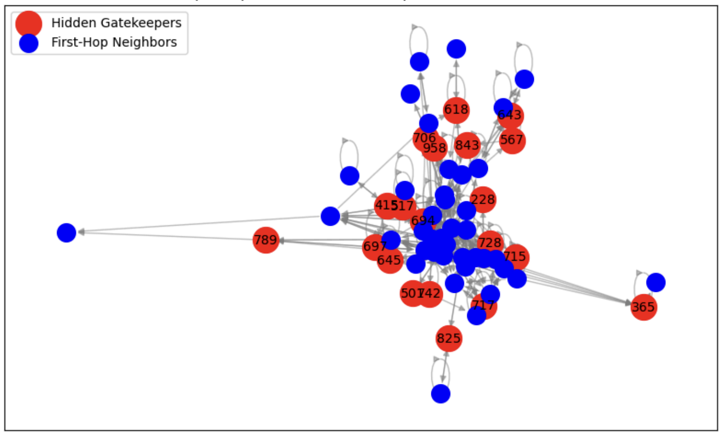
Who Are the Hidden Gatekeepers Controlling Information Flow?

Some employees don’t send many emails, but they control critical pathways that connect different teams. These Hidden Gatekeepers act as communication bottlenecks—if they become unavailable, information flow may slow down or stop entirely. The network graph highlights these employees in red and their direct neighbours in blue, showing their role in linking key groups. The code first calculates betweenness centrality, identifying employees who frequently appear on the shortest paths between others. It then filters out employees with low out-degree, meaning they don’t send many emails but still control key connections. For Telex, AI can suggest backup communicators to prevent delays, ensuring smooth collaboration even if a Hidden Gatekeeper is absent



```
In [56]: # Compute betweenness centrality
         betweenness_centrality = nx.betweenness_centrality(G)

         # Compute out-degree (emails sent)
         out_degree = dict(G.out_degree())

         # Identify hidden gatekeepers: high betweenness, low out-degree
         hidden_gatekeepers = {node: betweenness_centrality[node] for node in G.nodes() if out_degree[node] < 5}  # Low out-degree

         # Get top 20 hidden gatekeepers
         top_gatekeepers = [x[0] for x in sorted(hidden_gatekeepers.items(), key=lambda x: x[1], reverse=True)[:20]]

         # Include immediate neighbors of the top gatekeepers for context
         subgraph_nodes = set(top_gatekeepers)
         for node in top_gatekeepers:
             subgraph_nodes.update(G.neighbors(node))  # Add first-degree connections

         # Create the subgraph
         G_sub = G.subgraph(subgraph_nodes)

         # Check if subgraph has nodes before plotting
         if len(G_sub.nodes) > 0:
             # Network Visualization
             plt.figure(figsize=(10, 6))
             pos = nx.spring_layout(G_sub)  # Positioning nodes

             # Draw subgraph edges
             nx.draw_networkx_edges(G_sub, pos, alpha=0.5, edge_color="gray")

             # Draw hidden gatekeepers (red)
             nx.draw_networkx_nodes(G_sub, pos, nodelist=top_gatekeepers, node_color="red", node_size=400, label="Hidden Gatekeepers")

             # Draw first-degree neighbors (blue)
             nx.draw_networkx_nodes(G_sub, pos, nodelist=set(G_sub.nodes()) - set(top_gatekeepers), node_color="blue", node_size=200, label="First-Hop Neighb

             # Add labels only to hidden gatekeepers
             nx.draw_networkx_labels(G_sub, pos, labels={n: str(n) for n in top_gatekeepers}, font_size=10, font_color="black")

             plt.legend()
             plt.title("Network Graph: Top 20 Hidden Gatekeepers with Direct Connections")
             plt.show()
         else:
             print("No connected subgraph found for the top gatekeepers.")
```

Network Graph: Top 20 Hidden Gatekeepers with Direct Connections



Conclusion & Recommendations for Telex

Question	Telex Feature Influenced	How It Helps Telex
Who are the "Information Routers" in the Network?	AI-Powered Message Routing	Identifies key employees who forward the most emails, allowing Telex to automate message distribution and ensure important information reaches the right teams faster.
Who Are the Most Well-Connected Employees?	Priority Messaging & Notifications	Finds employees who can spread messages the fastest, ensuring that urgent announcements reach high-impact individuals first for quicker response times.
Hidden Influencers: Employees Who Rarely Send Emails but Control Information Flow	Smart Notification System	Detects employees who appear in many discussions but don't actively send emails, helping Telex prioritize notifications so they stay informed without being overloaded.
Which Departments Have Centralized Communication?	Balanced Message Distribution & Workflow Automation	Identifies departments where a few individuals handle most emails, allowing Telex AI to recommend better distribution of communication tasks to prevent bottlenecks.
How Do Teams Naturally Cluster?	AI-Suggested Chat Groups & Team Channels	Uses real email interactions to group employees based on actual communication patterns, rather than formal department structures, ensuring more effective team collaboration.
Who Are the Hidden Gatekeepers Controlling Information Flow?	Backup Communicators & Fail-Safe Messaging	Identifies employees who control critical pathways but don't send many emails, enabling Telex to suggest backups to prevent communication breakdowns if they are unavailable.
Which employees act as "Silent Connectors," bridging different teams but rarely initiating communication?	Cross-Team Collaboration Enhancement	Detects employees who link different teams but don't actively communicate, helping Telex connect them to relevant discussions or suggest key updates to keep all teams aligned.