NETWORK ANALYSIS OF ORGANIZATIONAL EMAIL COMMUNICATION

LIBRARIES IMPORTATION

The necessary Python libraries were imported to facilitate data manipulation, network analysis, and visualization. Pandas was used for handling structured datasets, allowing for efficient data manipulation. NetworkX enabled the creation and analysis of graphs, making it possible to study network structures. Matplotlib.pyplot provided basic visualization tools for plotting graphs and charts, while Seaborn enhanced the visualization process with statistical and aesthetically appealing plots. Together, these libraries made it easier to load, analyze, and visualize both tabular and network data efficiently.

```
#Instal and Import Required Libraries
import networkx as nx
import pandas as pd
import matplotlib.pyplot as plt
import random
import community as community_louvain  # Louvain method for community
detection
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, accuracy_score
```

DATA LOADING

The code loads two datasets related to email communication and department labels. The first dataset, `email_df`, contains records of email interactions, where each row represents an email sent from one individual to another. The second dataset, `department_df`, maps individuals to their respective departments, assigning each node a department label. After loading the data, the first few rows of both datasets are displayed to verify their structure and content.

```
# Load the email dataset
email_df = pd.read_csv("email-Eu-core [MConverter.eu].csv",
delimiter=",", header=None, names=["Sender", "Receiver"])

# Load department labels
department_df = pd.read_csv("email-Eu-core-department-labels
[MConverter.eu].csv", delimiter=",", header=None, names=["Node",
"Department"])

# Preview data
print(email_df.head())
print(department_df.head())
```

```
   Sender  Receiver
0       0         1
1       2         3
2       2         4
3       5         6
4       5         7
   Node  Department
0     0           1
1     1           1
2     2          21
3     3          21
4     4          21
```

##DATA EXPLORATORY

##Datasets Information

The code prints the number of email interactions and department nodes in the datasets. It first displays a label, "Dataset Information," and then calculates the total number of emails by counting the rows in `email_df`. Similarly, it determines the number of unique nodes in the department dataset by counting the rows in `department_df`.

```
print("\nDataset Information:")
print("Number of email:", email_df.shape[0])
print("Number of nodes in department:", department_df.shape[0])
```

```
Dataset Information:
Number of email: 25571
Number of nodes in department: 1005
```

Missing Values Check and Columns Clarity

The code renames columns for clarity and checks data types and missing values in both datasets, storing the results for further analysis.

```
# Rename columns for clarity
email_df.columns = ["Sender", "Receiver"]
department_df.columns = ["Node", "Department"]

# Check data types and missing values
email_info = email_df.info(), email_df.isnull().sum()
department_info = department_df.info(), department_df.isnull().sum()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25571 entries, 0 to 25570
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
```

```
 0   Sender    25571 non-null  int64
 1   Receiver  25571 non-null  int64
dtypes: int64(2)
memory usage: 399.7 KB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1005 entries, 0 to 1004
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Node        1005 non-null   int64
 1   Department  1005 non-null   int64
dtypes: int64(2)
memory usage: 15.8 KB
```

BASIC NETWORK ANALYSIS

The code constructs a directed graph from the email dataset, where senders and receivers form network connections. It then assigns department labels to nodes as attributes using a dictionary. Finally, it prints key graph metrics, including the number of nodes, edges, and the average degree, which represents the average number of connections per node.

```
# Create graph from email network
G = nx.from_pandas_edgelist(email_df, source="Sender",
target="Receiver", create_using=nx.DiGraph())

# Add department labels as node attributes
dept_dict = department_df.set_index("Node")["Department"].to_dict()
nx.set_node_attributes(G, dept_dict, "Department")

# Check basic graph info
print(f"Number of Nodes: {G.number_of_nodes()}")
print(f"Number of Edges: {G.number_of_edges()}")
print(f"avg_degree: {sum(dict(G.degree()).values()) /
G.number_of_nodes()}")
```

```
Number of Nodes: 1005
Number of Edges: 25571
avg_degree: 50.88756218905473
```

```
# Basic graph statistics
num_nodes = G.number_of_nodes()
num_edges = G.number_of_edges()
density = nx.density(G)

print(f"Network Statistics:")
print(f"Total Employees (Nodes): {num_nodes}")
print(f"Total Email Interactions (Edges): {num_edges}")
print(f"Network Density: {density:.4f}")
```

```
Network Statistics:
Total Employees (Nodes): 1005
Total Email Interactions (Edges): 25571
Network Density: 0.0253
```

Number of Emails Sent and Received by Department

The code analyzes email communication between departments by converting department labels into a dictionary for quick lookup. It then iterates through the email dataset, counting the number of outgoing and incoming emails for each department. These counts are stored in dictionaries and later converted into a DataFrame, where each department is listed alongside the number of emails sent and received. The DataFrame is sorted in descending order based on emails sent, and the results are displayed.

```
import pandas as pd

# Convert department labels into a dictionary for quick lookup
dept_dict = department_df.set_index("Node")["Department"].to_dict()

# Create dictionaries to track outgoing and incoming emails per
department
outgoing_emails = {}
incoming_emails = {}

for sender, receiver in email_df.itertuples(index=False):
    if sender in dept_dict and receiver in dept_dict:
        sender_dept = dept_dict[sender]
        receiver_dept = dept_dict[receiver]

        # Count outgoing emails per department
        outgoing_emails[sender_dept] =
outgoing_emails.get(sender_dept, 0) + 1

        # Count incoming emails per department
        incoming_emails[receiver_dept] =
incoming_emails.get(receiver_dept, 0) + 1

# Convert dictionaries to DataFrame
df_dept_email_counts = pd.DataFrame({
    "Department":
list(set(outgoing_emails.keys()).union(set(incoming_emails.keys()))),
    "Emails Sent": [outgoing_emails.get(dept, 0) for dept in
set(outgoing_emails.keys()).union(set(incoming_emails.keys()))],
    "Emails Received": [incoming_emails.get(dept, 0) for dept in
set(outgoing_emails.keys()).union(set(incoming_emails.keys()))]
})

# Sort departments based on emails sent
df_dept_email_counts = df_dept_email_counts.sort_values(by="Emails
```

```
Sent", ascending=False)

# Display results
print("Number of Emails Sent and Received by Department")
print(df_dept_email_counts)
```

```
Number of Emails Sent and Received by Department
    Department  Emails Sent  Emails Received
4            4         2652             2700
36          36         2334             1905
14          14         2100             2273
21          21         1354             1395
7            7         1222             1252
10          10         1164             1207
1            1         1147             1326
15          15         1093             1097
13          13          918              942
0            0          910              986
34          34          798              759
11          11          746              856
19          19          695              685
17          17          684              767
22          22          597              669
5            5          581              615
35          35          569              527
9            9          530              556
26          26          508              368
16          16          475              499
38          38          467              405
8            8          466              493
20          20          442              428
25          25          434              401
37          37          378              403
23          23          354              268
32          32          318              214
6            6          308              271
28          28          265              241
3            3          197              255
2            2          172              192
40          40          137               83
27          27          131              105
31          31          119               74
29          29           80              109
12          12           76               74
39          39           74               65
24          24           52               54
30          30           21               29
41          41            3               14
18          18            0                6
33          33            0                3
```

Clustering Coefficient for Each Node

The code calculates the clustering coefficient for each node in the network, identifying the top 10 with the highest values. It then converts the results into a DataFrame for better visualization and displays it.

```
from IPython.display import display

# Compute clustering coefficient for each node
clustering_coefficients = nx.clustering(G.to_undirected())

# Identify top 10 nodes with the highest clustering coefficients
top_clustering_nodes = sorted(clustering_coefficients.items(),
key=lambda x: x[1], reverse=True)[:10]

# Convert to DataFrame for visualization
df_clustering = pd.DataFrame(top_clustering_nodes, columns=["Node",
"Clustering Coefficient"])

# Display results
display(df_clustering)
```


```
{"summary":"{\n  \"name\": \"df_clustering\",\n  \"rows\": 10,\n
\"fields\": [\n    {\n      \"column\": \"Node\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
110,\n        \"min\": 348,\n        \"max\": 628,\n
\"num_unique_values\": 10,\n        \"samples\": [\n          625,\n
382,\n          439\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Clustering Coefficient\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.0,\n        \"min\": 1.0,\n
\"max\": 1.0,\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          1.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df_clustering"}
```


Average Clustering Coefficient Per Department

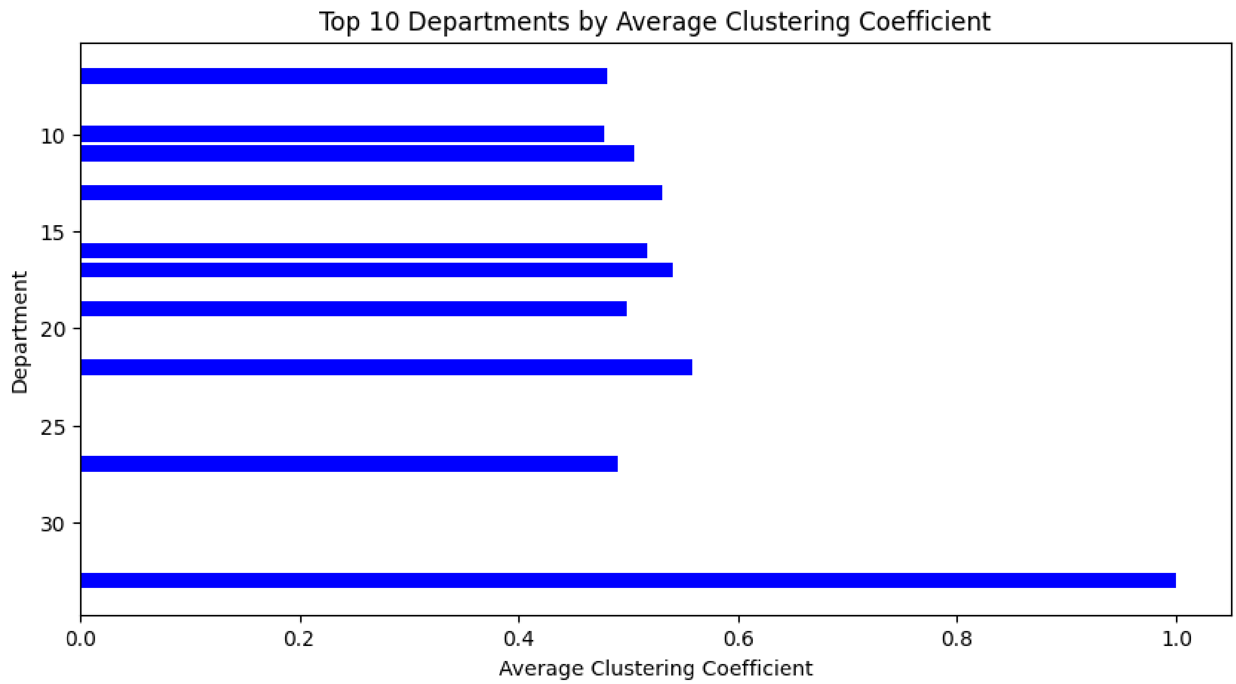
The code calculates the average clustering coefficient for each department, selects the top 10 departments with the highest values, and displays the results. It optionally saves the data to a CSV file and visualizes it using a bar chart for better interpretation.

```
# Compute average clustering coefficient per department
department_df["Clustering Coefficient"] =
department_df["Node"].map(clustering_coefficients)
dept_clustering = department_df.groupby("Department")["Clustering
Coefficient"].mean().reset_index()

# Select top 10 departments by clustering coefficient
```

```python
top_10_dept_clustering = dept_clustering.nlargest(10, "Clustering
Coefficient")

# Display results
display(top_10_dept_clustering)

# Optional: Save results to CSV
top_10_dept_clustering.to_csv("top_10_clustering_per_department.csv",
index=False)
print("Data saved as top_10_clustering_per_department.csv")

# Optional: Visualize results with a bar chart
plt.figure(figsize=(10, 5))
plt.barh(top_10_dept_clustering["Department"],
top_10_dept_clustering["Clustering Coefficient"], color="blue")
plt.xlabel("Average Clustering Coefficient")
plt.ylabel("Department")
plt.title("Top 10 Departments by Average Clustering Coefficient")
plt.gca().invert_yaxis()  # Invert y-axis for better readability
plt.show()
```


```
{"summary":"{\n  \"name\": \"top_10_dept_clustering\",\n  \"rows\":
10,\n  \"fields\": [\n    {\n      \"column\": \"Department\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
8,\n        \"min\": 7,\n        \"max\": 33,\n
\"num_unique_values\": 10,\n        \"samples\": [\n          7,\n
22,\n          11\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Clustering Coefficient\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.15671862552538737,\n
\"min\": 0.47850187670230976,\n        \"max\": 1.0,\n
\"num_unique_values\": 10,\n        \"samples\": [\n
0.4805026357621867,\n          0.557937158540009,\n
0.5059163698310525\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"top_10_dept_clustering"}
```


```
Data saved as top_10_clustering_per_department.csv
```



DEEP RESEARCH QUESTIONS

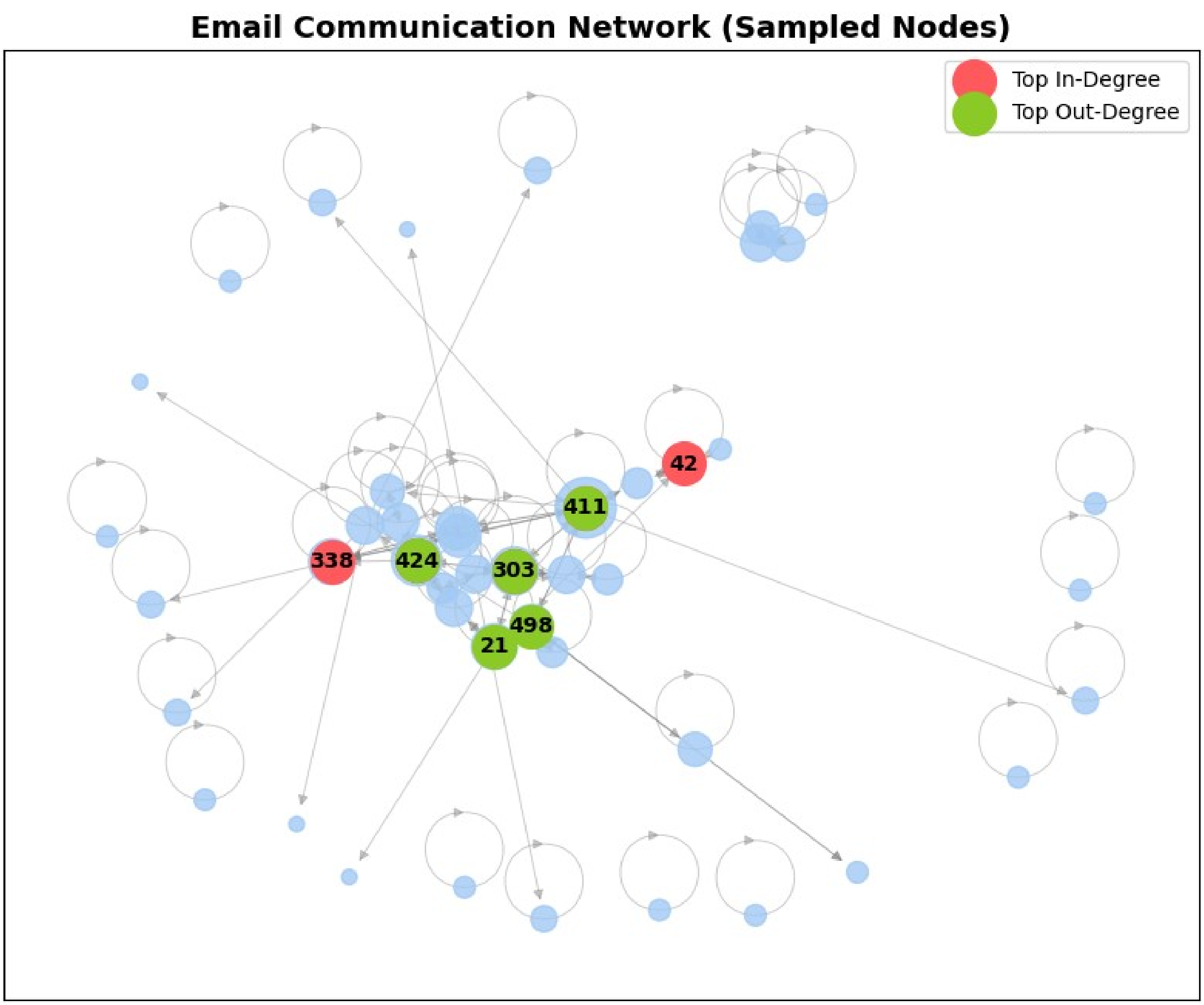
1. Which individuals have the highest in-degree and out-degree in the network? What does this reveal about key communicators?

The code creates a smaller email network, identifies top communicators, and visualizes the connections with highlighted key nodes.

```
# Create a directed graph
G = nx.DiGraph()
G.add_edges_from(email_df.values)

# Compute in-degree and out-degree centrality
in_degree_centrality = dict(G.in_degree())  # Number of received emails
out_degree_centrality = dict(G.out_degree())  # Number of sent emails

# Convert to DataFrame for easier analysis
centrality_df = pd.DataFrame({
    "Node": list(G.nodes()),
    "In-Degree": [in_degree_centrality[node] for node in G.nodes()],
    "Out-Degree": [out_degree_centrality[node] for node in G.nodes()]
})

# Identify top communicators
top_in_degree_nodes = centrality_df.sort_values(by="In-Degree",
```

```
ascending=False).head(5)["Node"].tolist()
top_out_degree_nodes = centrality_df.sort_values(by="Out-Degree",
ascending=False).head(5)["Node"].tolist()

#print("Top 5 Individuals with Highest In-Degree (Most Emails
Received):")
#print(top_in_degree)

# Create a directed graph
G = nx.DiGraph()
G.add_edges_from(email_df.values)

# Reduce network size for readability (Sample 50 nodes)
sampled_nodes = random.sample(list(G.nodes()), min(50,
len(G.nodes())))
H = G.subgraph(sampled_nodes)  # Extract subgraph

# Compute in-degree and out-degree centrality
in_degree_centrality = dict(H.in_degree())
out_degree_centrality = dict(H.out_degree())

# Identify top communicators
top_in_degree_nodes = sorted(in_degree_centrality,
key=in_degree_centrality.get, reverse=True)[:5]
top_out_degree_nodes = sorted(out_degree_centrality,
key=out_degree_centrality.get, reverse=True)[:5]

# Node size based on degree centrality
node_sizes = [H.degree(n) * 50 for n in H.nodes()]  # Scale node sizes

# Graph Visualization
plt.figure(figsize=(10, 8))
pos = nx.spring_layout(H, seed=42)  # Force-directed layout

# Draw regular nodes
nx.draw_networkx_nodes(H, pos, node_size=node_sizes, alpha=0.8,
node_color="#A1C9F4")  # Light Blue

# Highlight top communicators
nx.draw_networkx_nodes(H, pos, nodelist=top_in_degree_nodes,
node_size=400, node_color="#FF595E", label="Top In-Degree")  # Red
nx.draw_networkx_nodes(H, pos, nodelist=top_out_degree_nodes,
node_size=400, node_color="#8AC926", label="Top Out-Degree")  # Green

# Draw edges
nx.draw_networkx_edges(H, pos, alpha=0.4, edge_color="gray",
width=0.7)

# Labels for top communicators
labels = {node: str(node) for node in top_in_degree_nodes +
```

```
top_out_degree_nodes}
nx.draw_networkx_labels(H, pos, labels, font_size=10,
font_color="black", font_weight="bold")

plt.title("Email Communication Network (Sampled Nodes)", fontsize=14,
fontweight="bold")
plt.legend()
plt.show()
```



##2. Are there distinct clusters or communities in the email network? How do these groups interact with one another?

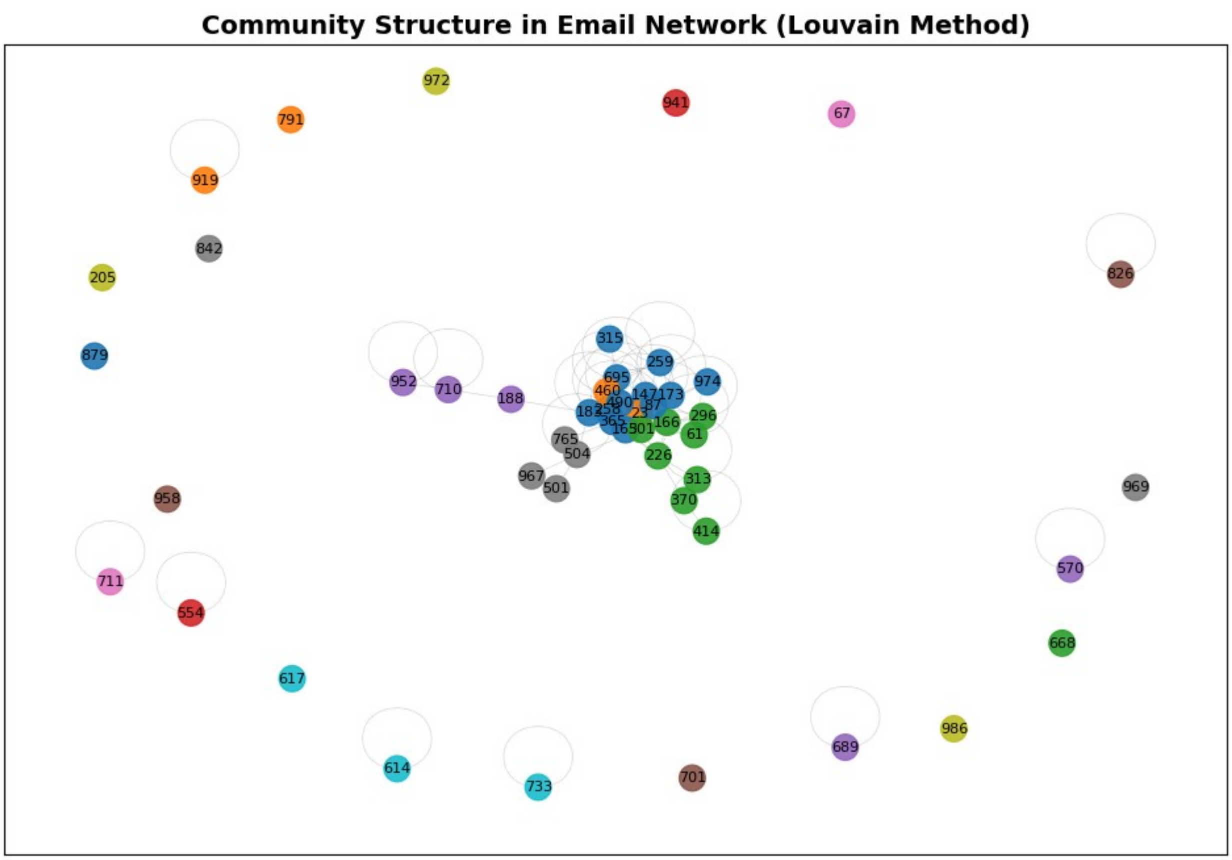
The code creates a smaller email network, calculates its density, and detects communities using the Louvain method. Nodes are assigned colors based on their communities, and the network is visualized with a force-directed layout, showing connections and group structures.

```
import networkx as nx
import pandas as pd
```

```
import random
import matplotlib.pyplot as plt
import community.community_louvain as community_louvain  # Correct
import

# Create a directed graph
G = nx.DiGraph()
G.add_edges_from(email_df.values)

# Reduce network size for readability (Sample 50 nodes)
sampled_nodes = random.sample(list(G.nodes()), min(50,
len(G.nodes())))
H = G.subgraph(sampled_nodes)  # Extract subgraph

# ------------------- NETWORK DENSITY ------------------- #
density = nx.density(H)
print(f"Network Density: {density:.4f}")  # Percentage of possible
connections realized

# ------------------- COMMUNITY DETECTION ------------------- #
# Convert to undirected graph for Louvain method
H_undirected = H.to_undirected()

# Compute communities using Louvain method
partition = community_louvain.best_partition(H_undirected)  # Fixed
import issue

# Assign community colors
unique_communities = list(set(partition.values()))
color_map = plt.cm.get_cmap("tab10", len(unique_communities))  #
Generate distinct colors
node_colors = [color_map(partition[node]) for node in
H_undirected.nodes()]

# ------------------- VISUALIZATION ------------------- #
plt.figure(figsize=(12, 8))
pos = nx.spring_layout(H_undirected, seed=42)  # Force-directed layout

# Draw nodes with community colors
nx.draw_networkx_nodes(H_undirected, pos, node_size=200,
node_color=node_colors, alpha=0.9)

# Draw edges
nx.draw_networkx_edges(H_undirected, pos, alpha=0.3,
edge_color="gray", width=0.5)

# Labels (Optional, for small graphs)
if len(H_undirected.nodes()) <= 50:
    nx.draw_networkx_labels(H_undirected, pos, font_size=8,
font_color="black")
```

```
plt.title("Community Structure in Email Network (Louvain Method)",
fontsize=14, fontweight="bold")
plt.show()
```

```
Network Density: 0.0469

<ipython-input-53-3074c7fb50eb>:28: MatplotlibDeprecationWarning: The
get_cmap function was deprecated in Matplotlib 3.7 and will be removed
in 3.11. Use ``matplotlib.colormaps[name]`` or
``matplotlib.colormaps.get_cmap()`` or ``pyplot.get_cmap()`` instead.
  color_map = plt.cm.get_cmap("tab10", len(unique_communities))  #
Generate distinct colors
```

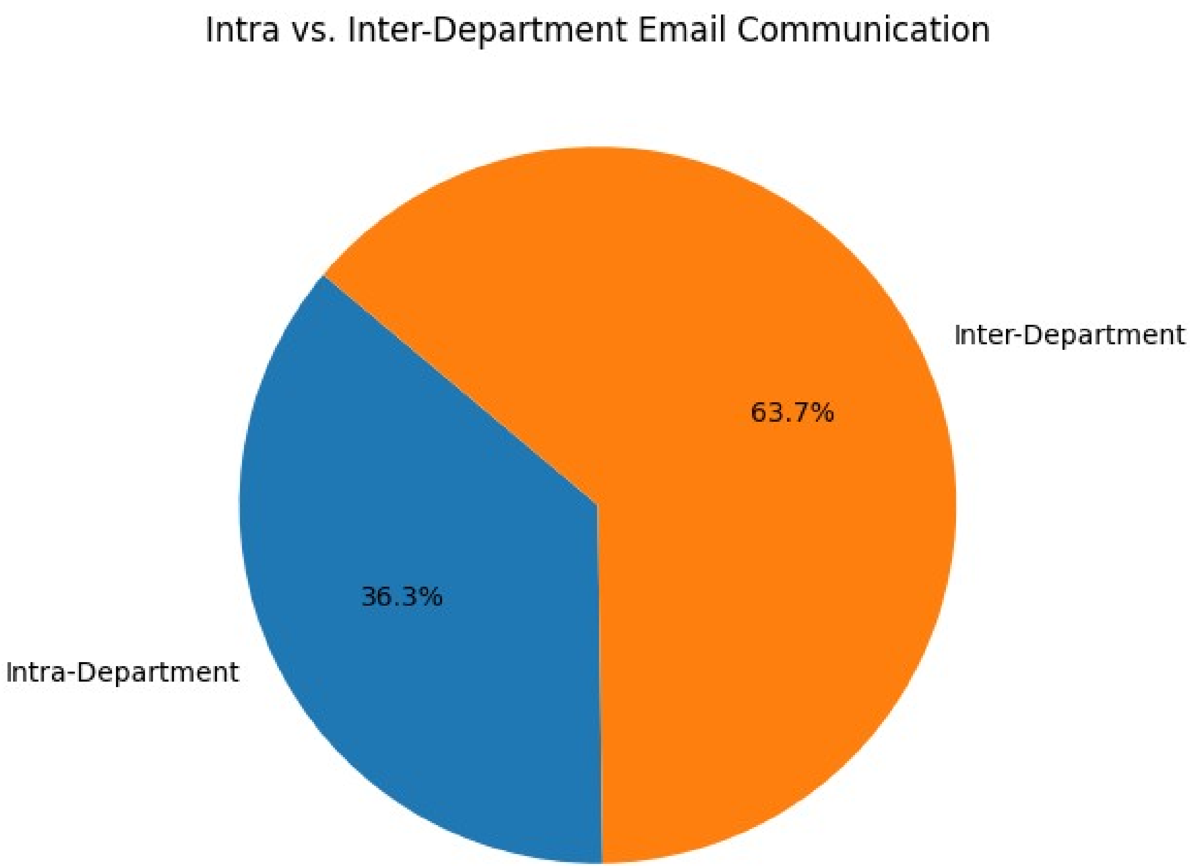


##3. Do employees communicate more within their department or across departments? The code categorizes emails as intra-department (within the same department) or inter-department (between different departments) using a department dictionary. It calculates and displays the percentage of each type and visualizes the distribution with a pie chart.

```
# Load department labels
department_df = pd.read_csv("email-Eu-core-department-labels
[MConverter.eu].csv", delimiter=",", header=None, names=["Node",
"Department"])
```

```python
# Convert to dictionary for quick lookup (Fix: Use "Node" instead of
"Employee")
dept_dict = department_df.set_index("Node")["Department"].to_dict()

# Count intra- and inter-department emails
intra_count = 0
inter_count = 0

for sender, receiver in email_df.itertuples(index=False):
    if sender in dept_dict and receiver in dept_dict:
        if dept_dict[sender] == dept_dict[receiver]:
            intra_count += 1  # Same department
        else:
            inter_count += 1  # Different department

# Calculate percentages
total_emails = intra_count + inter_count
intra_percent = (intra_count / total_emails) * 100
inter_percent = (inter_count / total_emails) * 100

# Print results
print(f"Intra-Department Emails: {intra_count} ({intra_percent:.2f}
%)")
print(f"Inter-Department Emails: {inter_count} ({inter_percent:.2f}
%)")

# Visualization
labels = ["Intra-Department", "Inter-Department"]
values = [intra_count, inter_count]
colors = ["#1f77b4", "#ff7f0e"]  # Blue and Orange

plt.figure(figsize=(6, 6))
plt.pie(values, labels=labels, autopct="%1.1f%%", colors=colors,
startangle=140)
plt.title("Intra vs. Inter-Department Email Communication")
plt.show()
```

```
Intra-Department Emails: 9287 (36.32%)
Inter-Department Emails: 16284 (63.68%)
```

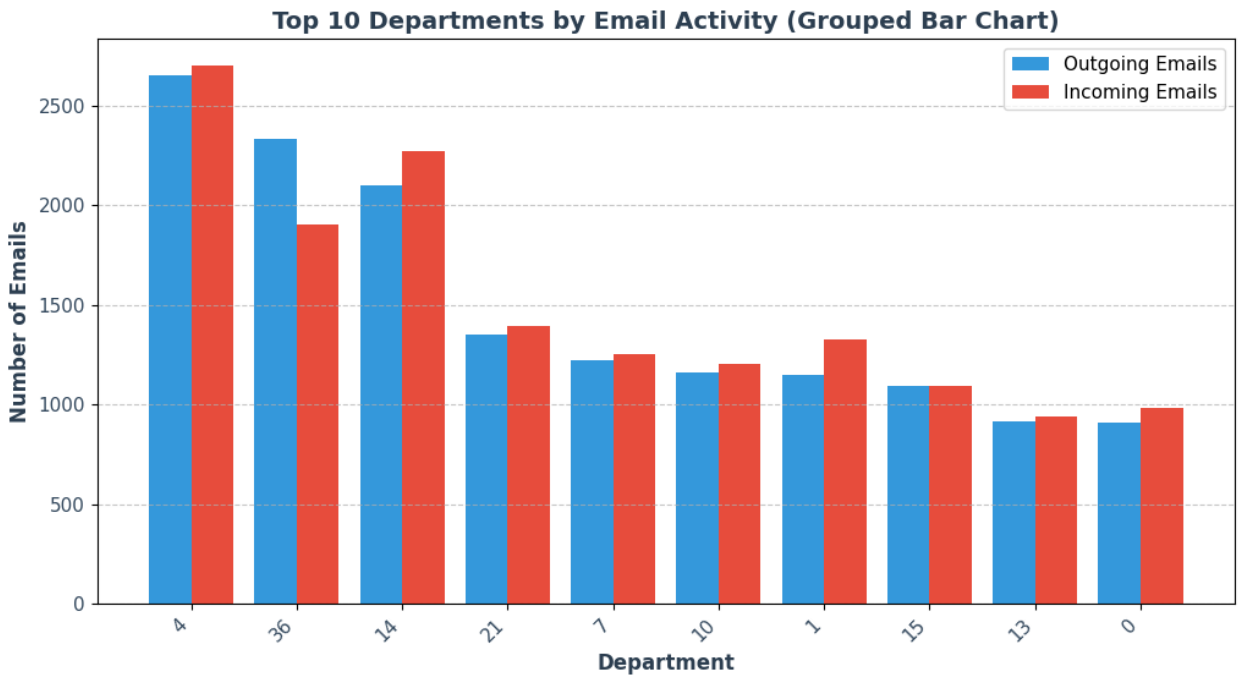


##4. Which departments have the highest number of outgoing emails? How does this compare to their incoming emails? The code analyzes email activity by department, counting outgoing and incoming emails. It then selects the top 10 departments based on outgoing emails and visualizes the data using a grouped bar chart, where outgoing and incoming emails are represented with distinct colors for clarity.

```
import seaborn as sns
import numpy as np

# Convert to dictionary for quick lookup
dept_dict = department_df.set_index("Node")["Department"].to_dict()

# Create dictionaries to track outgoing and incoming emails per
department
outgoing_emails = {}
incoming_emails = {}

for sender, receiver in email_df.itertuples(index=False):
    if sender in dept_dict and receiver in dept_dict:
        sender_dept = dept_dict[sender]
```

```
        receiver_dept = dept_dict[receiver]

        # Count outgoing emails per department
        outgoing_emails[sender_dept] =
outgoing_emails.get(sender_dept, 0) + 1

        # Count incoming emails per department
        incoming_emails[receiver_dept] =
incoming_emails.get(receiver_dept, 0) + 1

# Convert dictionaries to DataFrame
df_dept_emails = pd.DataFrame({
    "Department": list(outgoing_emails.keys()),
    "Outgoing Emails": [outgoing_emails[dept] for dept in
outgoing_emails],
    "Incoming Emails": [incoming_emails.get(dept, 0) for dept in
outgoing_emails]  # Fill missing departments with 0
})

# Sort by outgoing emails and select the top 10 departments
df_top10 = df_dept_emails.nlargest(10, "Outgoing Emails")

# Define position of bars on X-axis
x = np.arange(len(df_top10["Department"]))
width = 0.4  # Width of bars

# Define catchy colors
colors = ["#3498db", "#e74c3c"]  # Blue & Red

# Visualization - Grouped Bar Chart
plt.figure(figsize=(12, 6))
plt.bar(x - width/2, df_top10["Outgoing Emails"], width,
label="Outgoing Emails", color=colors[0])
plt.bar(x + width/2, df_top10["Incoming Emails"], width,
label="Incoming Emails", color=colors[1])

# Labeling
plt.xlabel("Department", fontsize=12, fontweight="bold",
color="#2c3e50")
plt.ylabel("Number of Emails", fontsize=12, fontweight="bold",
color="#2c3e50")
plt.title("Top 10 Departments by Email Activity (Grouped Bar Chart)",
fontsize=14, fontweight="bold", color="#2c3e50")
plt.xticks(x, df_top10["Department"], rotation=45, ha="right",
fontsize=11, color="#34495e")  # Rotate labels
plt.yticks(fontsize=11, color="#34495e")
plt.legend(fontsize=11)

# Add gridlines for better readability
plt.grid(axis="y", linestyle="--", alpha=0.7)
```

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

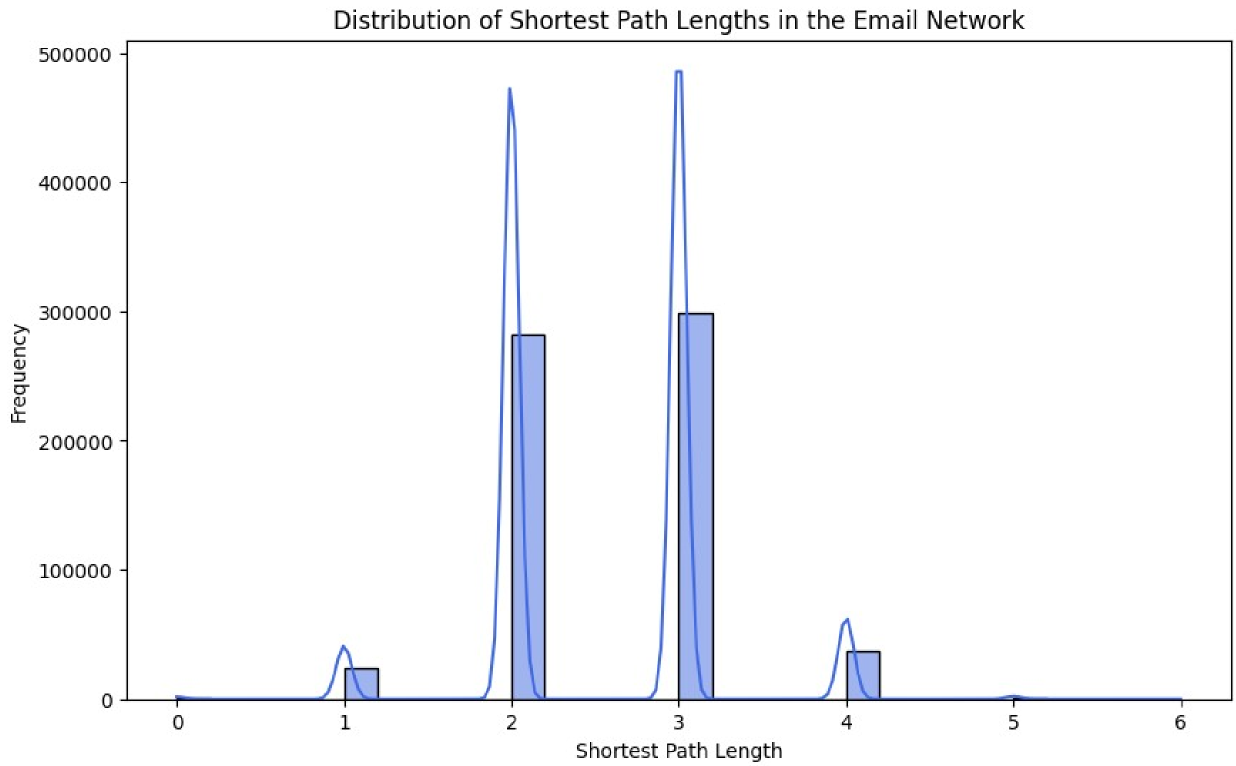


##5. What is the average shortest path between any two individuals? How efficiently does information flow through the network? The code constructs a directed graph from the email dataset and checks if it is strongly connected. If so, it calculates the average shortest path length for the entire network; otherwise, it focuses on the largest strongly connected component (LSCC). It then computes and visualizes the distribution of shortest path lengths using a histogram to analyze communication efficiency.

```
import networkx as nx
import numpy as np

# Create a directed graph from the email dataset
G = nx.DiGraph()
G.add_edges_from(email_df.itertuples(index=False, name=None))

# Check if the graph is strongly connected (i.e., every node can reach
every other node)
if nx.is_strongly_connected(G):
    # Compute the average shortest path length for the entire network
    avg_shortest_path = nx.average_shortest_path_length(G)
    print(f"Average Shortest Path Length (Strongly Connected Graph):
{avg_shortest_path:.2f}")

else:
    # If the graph is not strongly connected, work with the largest
```

```
strongly connected component (LSCC)
    largest_scc = max(nx.strongly_connected_components(G), key=len)
    G_scc = G.subgraph(largest_scc)

    avg_shortest_path_scc = nx.average_shortest_path_length(G_scc)
    print(f"Average Shortest Path Length (Largest Strongly Connected
Component): {avg_shortest_path_scc:.2f}")

# Compute shortest path distribution
shortest_paths = []
for component in nx.strongly_connected_components(G):
    subgraph = G.subgraph(component)
    try:
        path_lengths =
dict(nx.all_pairs_shortest_path_length(subgraph))
        for lengths in path_lengths.values():
            shortest_paths.extend(lengths.values())
    except:
        pass

# Visualizing the shortest path distribution
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 6))
sns.histplot(shortest_paths, bins=30, kde=True, color="royalblue")
plt.xlabel("Shortest Path Length")
plt.ylabel("Frequency")
plt.title("Distribution of Shortest Path Lengths in the Email
Network")
plt.show()
```

```
Average Shortest Path Length (Largest Strongly Connected Component):
2.55
```



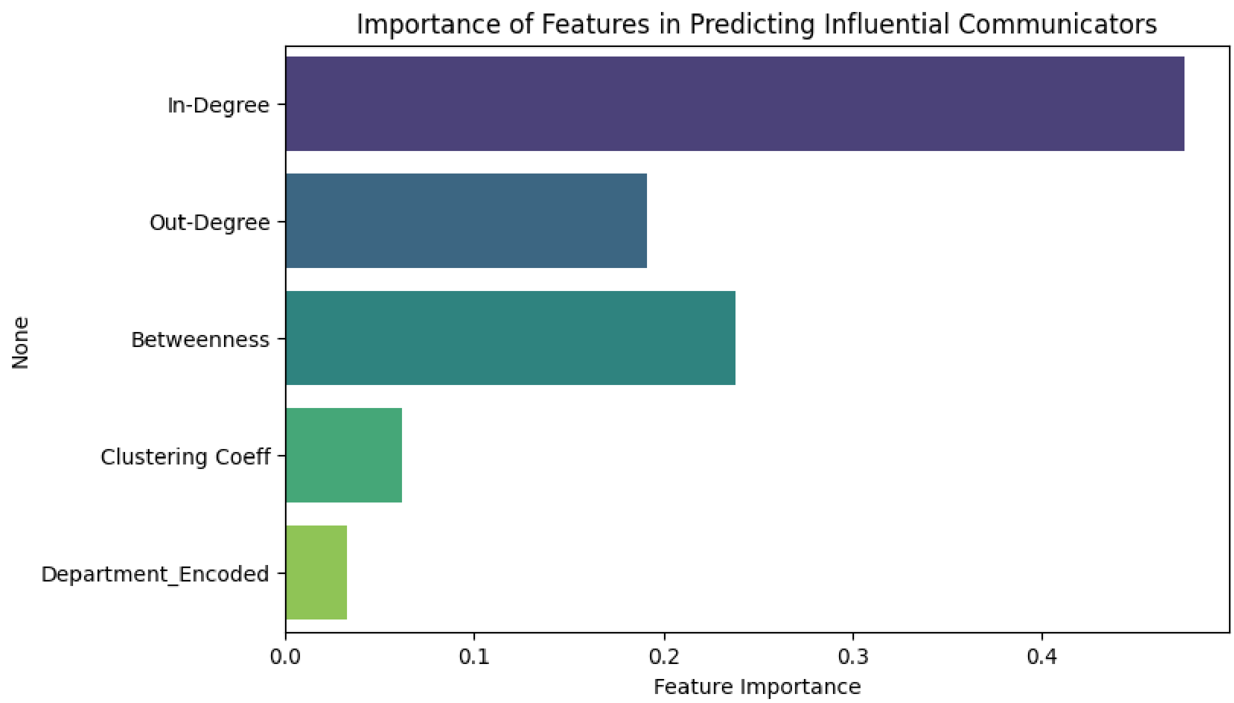
##6. Can we predict which individuals are likely to become key communicators based on their email activity and department affiliation? The code constructs a directed email network, computes key network metrics, and identifies influential individuals using PageRank. A Random Forest model predicts influencers based on these metrics, and feature importance is visualized.

```
# Create a directed graph from the email dataset
G = nx.DiGraph()
G.add_edges_from(email_df.itertuples(index=False, name=None))

# Compute network features
in_degree = dict(G.in_degree())  # Emails received
out_degree = dict(G.out_degree())  # Emails sent
betweenness = nx.betweenness_centrality(G)
pagerank = nx.pagerank(G)
clustering_coeff = nx.clustering(G.to_undirected())

# Department Mapping
dept_dict = department_df.set_index("Node")["Department"].to_dict()

# Create DataFrame
df_features = pd.DataFrame({
    "Node": list(G.nodes()),
    "In-Degree": [in_degree[node] for node in G.nodes()],
    "Out-Degree": [out_degree[node] for node in G.nodes()],
    "Betweenness": [betweenness[node] for node in G.nodes()],
    "PageRank": [pagerank[node] for node in G.nodes()],
```

```
    "Clustering Coeff": [clustering_coeff.get(node, 0) for node in
G.nodes()],
    "Department": [dept_dict.get(node, "Unknown") for node in
G.nodes()]
})

# Define "Influential" (Top 10% PageRank)
threshold = np.percentile(df_features["PageRank"], 90)
df_features["Influential"] = (df_features["PageRank"] >=
threshold).astype(int)  # 1 = Influential, 0 = Not

# Encode Department
label_encoder = LabelEncoder()
df_features["Department_Encoded"] =
label_encoder.fit_transform(df_features["Department"])

# Select Features and Target
X = df_features[["In-Degree", "Out-Degree", "Betweenness", "Clustering
Coeff", "Department_Encoded"]]
y = df_features["Influential"]

# Train/Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Train a Random Forest Classifier
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)

# Predictions
y_pred = clf.predict(X_test)

# Evaluation
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

# Feature Importance Plot
plt.figure(figsize=(8, 5))
sns.barplot(x=clf.feature_importances_, y=X.columns,
palette="viridis")
plt.xlabel("Feature Importance")
plt.title("Importance of Features in Predicting Influential
Communicators")
plt.show()
```

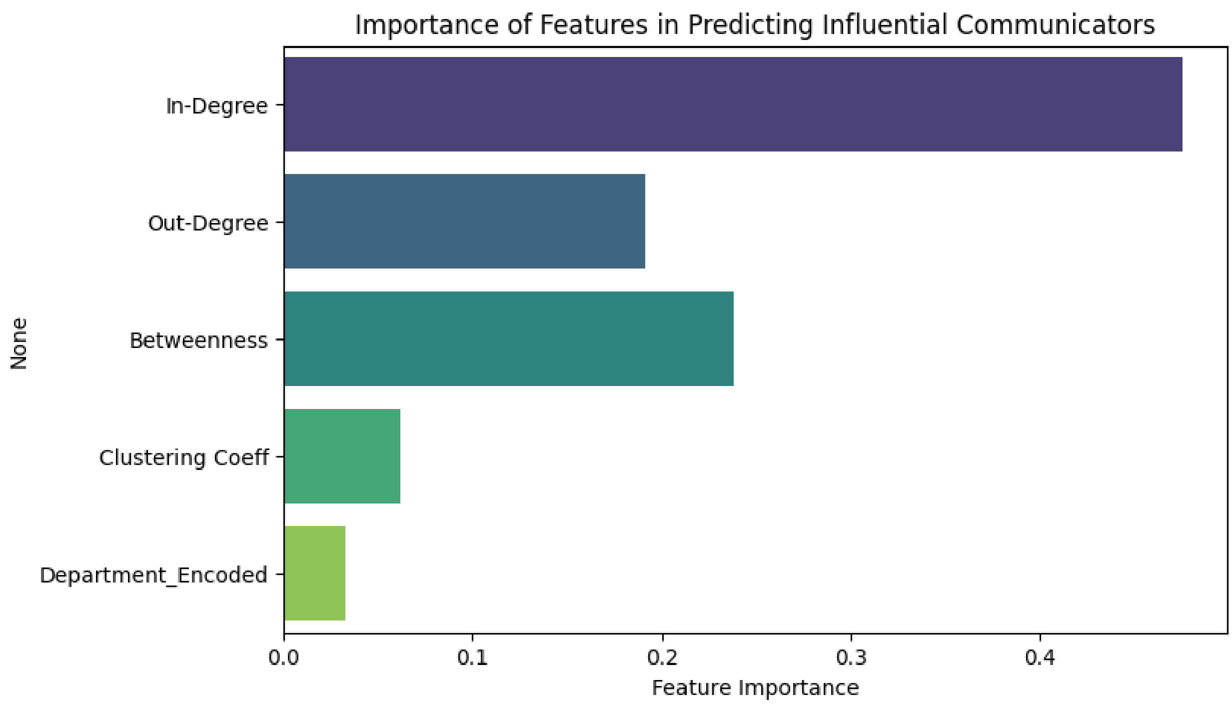
```
Accuracy: 0.9800995024875622
              precision    recall  f1-score   support

           0       0.99      0.98      0.99       180
           1       0.87      0.95      0.91        21
```

```
    accuracy                           0.98       201
   macro avg       0.93      0.97      0.95       201
weighted avg       0.98      0.98      0.98       201


<ipython-input-60-b7e99a16ee3a>:54: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `y` variable to `hue` and set
`legend=False` for the same effect.

  sns.barplot(x=clf.feature_importances_, y=X.columns,
palette="viridis")
```




```
Accuracy: 0.9800995024875622
              precision    recall  f1-score   support

           0       0.99      0.98      0.99       180
           1       0.87      0.95      0.91        21

    accuracy                           0.98       201
   macro avg       0.93      0.97      0.95       201
weighted avg       0.98      0.98      0.98       201


<ipython-input-61-b7e99a16ee3a>:54: FutureWarning:
```

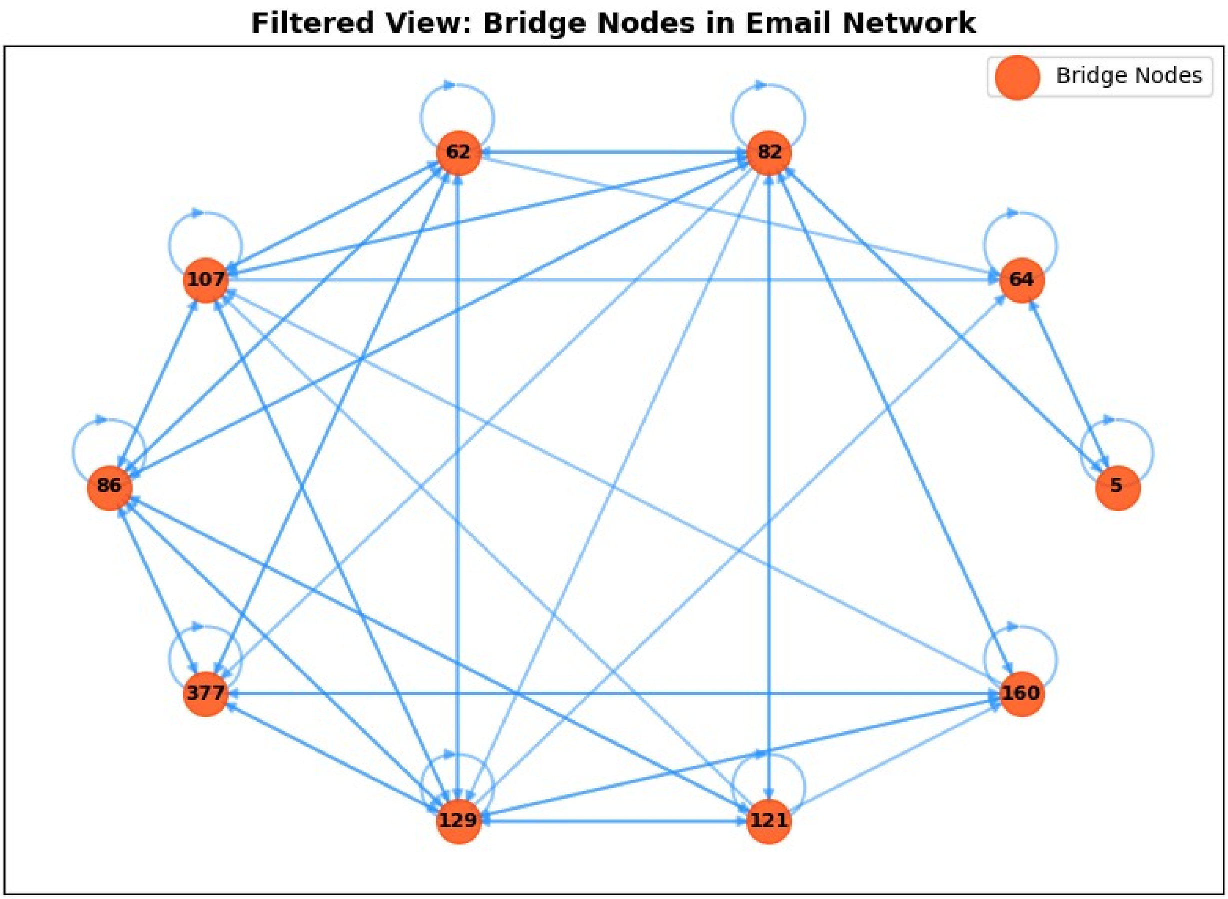
```
Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `y` variable to `hue` and set
`legend=False` for the same effect.

  sns.barplot(x=clf.feature_importances_, y=X.columns,
palette="viridis")
```



##7. Which individuals serve as "bridges" between different groups? How critical are they for communication across departments? The code constructs a directed email network, calculates betweenness centrality to identify the top 10 bridge nodes, and visualizes their direct connections using a circular layout. It highlights key intermediaries in communication flow, with edges in blue and nodes in orange-red.

```
# Create a directed graph from the email dataset
G = nx.DiGraph()
G.add_edges_from(email_df.itertuples(index=False, name=None))

# Compute betweenness centrality and select top bridge nodes
betweenness_centrality = nx.betweenness_centrality(G)
df_betweenness = pd.DataFrame(betweenness_centrality.items(),
columns=["Node", "Betweenness Centrality"])
df_betweenness = df_betweenness.sort_values(by="Betweenness
Centrality", ascending=False)
top_bridge_nodes = df_betweenness.head(10)["Node"].tolist()

# Filter edges to show only direct connections among top bridge nodes
filtered_edges = [(u, v) for u, v in G.edges() if u in
```

```
top_bridge_nodes and v in top_bridge_nodes]
G_filtered = nx.DiGraph()
G_filtered.add_edges_from(filtered_edges)

# Define layout (Circular layout for better clarity)
plt.figure(figsize=(10, 7))
pos = nx.circular_layout(G_filtered)

# Draw nodes
nx.draw_networkx_nodes(G_filtered, pos, node_size=400, alpha=0.8,
node_color="orangered", label="Bridge Nodes")

# Draw edges
nx.draw_networkx_edges(G_filtered, pos, alpha=0.5,
edge_color="dodgerblue", width=1.5, arrowsize=10)

# Labels for bridge nodes
labels = {node: str(node) for node in top_bridge_nodes}
nx.draw_networkx_labels(G_filtered, pos, labels, font_size=9,
font_color="black", font_weight="bold")

# Title and legend
plt.title("Filtered View: Bridge Nodes in Email Network", fontsize=13,
fontweight="bold")
plt.legend()
plt.show()
```

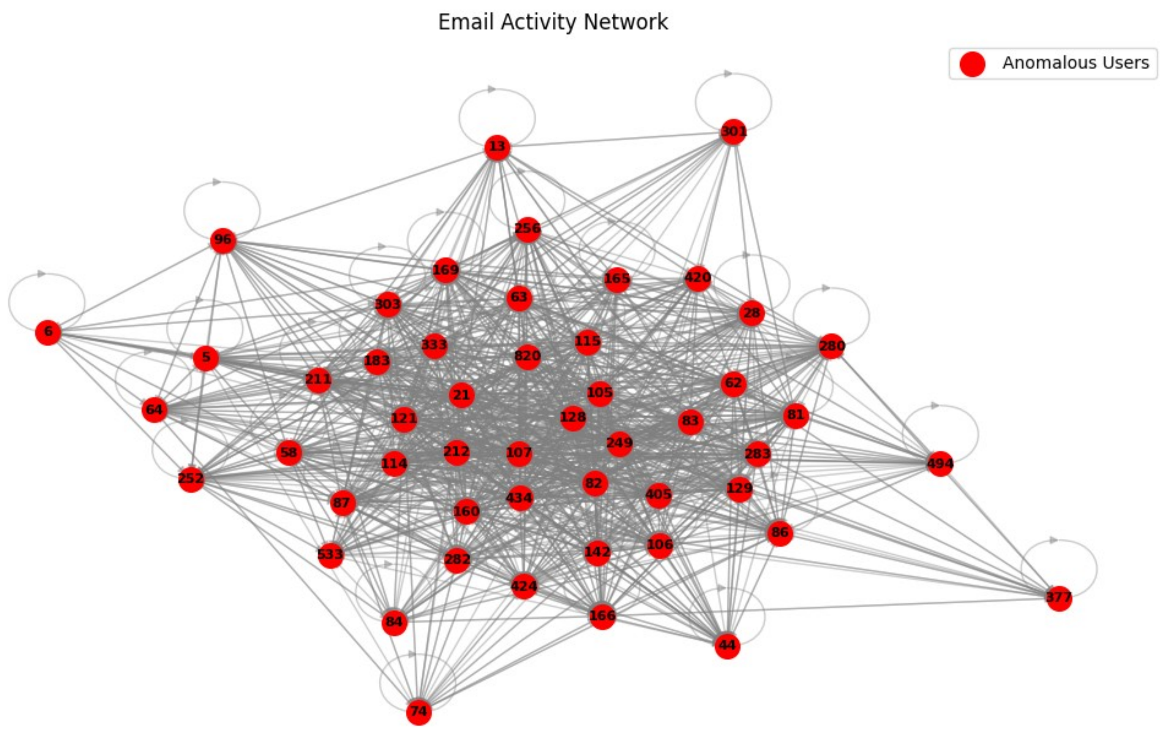


##8. Are there individuals who exhibit unusual email activity? Could these anomalies indicate urgent business situations or potential security threats? The code analyzes email activity by counting sent and received emails per user, then detects anomalies using the Isolation Forest algorithm. Users with unusual activity patterns are flagged as anomalies. The graph visualizes top active users and anomalies, highlighting anomalous users in red.

```
import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt
from sklearn.ensemble import IsolationForest

# Compute email activity per user
email_activity = email_df["Sender"].value_counts().to_frame("Emails 
Sent").join(
    email_df["Receiver"].value_counts().to_frame("Emails Received"), 
how="outer"
).fillna(0)
email_activity["Total Activity"] = email_activity.sum(axis=1)

# Detect anomalies using Isolation Forest
iso_forest = IsolationForest(contamination=0.05, random_state=42)
email_activity["Anomaly"] =
```

```
iso_forest.fit_predict(email_activity[["Total Activity"]])

# Extract anomalous users
anomalous_users = email_activity[email_activity["Anomaly"] == -
1].index.tolist()

# Filter graph to show only top active nodes and anomalous users
top_active_users = email_activity.nlargest(20, "Total
Activity").index.tolist()
filtered_nodes = set(top_active_users + anomalous_users)
filtered_edges = email_df[email_df["Sender"].isin(filtered_nodes) &
email_df["Receiver"].isin(filtered_nodes)]

# Create and visualize the filtered graph
G = nx.from_pandas_edgelist(filtered_edges, "Sender", "Receiver",
create_using=nx.DiGraph())
pos = nx.spring_layout(G, seed=42)

plt.figure(figsize=(10, 6))
nx.draw(G, pos, node_size=40, alpha=0.4, node_color="lightgray",
edge_color="gray")
nx.draw_networkx_nodes(G, pos, nodelist=anomalous_users,
node_size=200, node_color="red", label="Anomalous Users")
nx.draw_networkx_labels(G, pos, {node: node for node in
anomalous_users}, font_size=8, font_weight="bold")

plt.title("Email Activity Network")
plt.legend()
plt.show()

print("Anomalous Users:", anomalous_users)
```

```
/usr/local/lib/python3.11/dist-packages/IPython/core/
pylabtools.py:151: UserWarning: Creating legend with loc="best" can be
slow with large amounts of data.
  fig.canvas.print_figure(bytes_io, **kw)
```




```
Anomalous Users: [5, 6, 13, 21, 28, 44, 58, 62, 63, 64, 74, 81, 82,
83, 84, 86, 87, 96, 105, 106, 107, 114, 115, 121, 128, 129, 142, 160,
165, 166, 169, 183, 211, 212, 249, 252, 256, 280, 282, 283, 301, 303,
333, 377, 405, 420, 424, 434, 494, 533, 820]
```