**Data Preparation**

**Task 1: Load the data into a Colab Notebook using an appropriate method.**

We selected 10 random json dataset from each of the years 2020, 2021, 2022. Which were stored in ‘deepl\_translated\_jabber/random10’ of our drive.

**Code:**

**import shutil**

**import random**

**data\_dir = '/content/drive/My Drive/deepl\_translated\_jabber'**

**def select\_files(year):**

**year\_dir = os.path.join(data\_dir, year)**

**files = os.listdir(year\_dir)**

**selected\_files = random.sample(files, 10)**

**return [os.path.join(year\_dir, file) for file in selected\_files]**

**selected\_files\_dir = os.path.join(data\_dir, 'random10')**

**if not os.path.exists(selected\_files\_dir):**

**os.makedirs(selected\_files\_dir)**

**for year in ['2020', '2021', '2022']:**

**selected\_files = select\_files(year)**

**year\_selected\_files\_dir = os.path.join(selected\_files\_dir, year)**

**if not os.path.exists(year\_selected\_files\_dir):**

**os.makedirs(year\_selected\_files\_dir)**

**for file in selected\_files:**

**shutil.copy(file, year\_selected\_files\_dir)**

**print("Selected files have been stored in the following directories:")**

**print(f"2020: {os.path.join(selected\_files\_dir, '2020')}")**

**print(f"2021: {os.path.join(selected\_files\_dir, '2021')}")**

**print(f"2022: {os.path.join(selected\_files\_dir, '2022')}")**

Initially we found that there were some issues on those datasets. We should note that the English translations were done using Google Translate, which is not the same as being translated by expert translators. We had to re-format the json files so that we could parse the given json files (Kostadinov, 2022). The format of these JSON files was that each JSON object was not separated by a comma and was instead placed on a new line without any delimiters. This made it difficult to parse the entire file as a single JSON object, leading to JSON decoding errors. So we needed to preprocess those files to fix the issue. To address this issue, we first stripped the data of each JSON file and then replaced it with placeholders for preprocessing according to the documentation (Python Software Foundation, n.d.). After completing the necessary fixes, we stored the corrected files in the 'preprocessedData' folder on our drive.

**Code:**

**def fix\_json\_format(file\_path):**

**with open(file\_path, 'r') as f:**

**data = f.read().strip()**

**fixed\_data = '[' + data.replace('}\n{', '},{') + ']'**

**with open(file\_path, 'w') as f:**

|  |  |
| --- | --- |
| Before (Invalid) | After (Valid) |
| {  "ts": "2022-01-09T11:27:55.712408",  "from": "tramp@q3mcco35auwcstmt.onion",  "to": "pumba@q3mcco35auwcstmt.onion",  "body": "yes"  }  {  "ts": "2022-01-09T11:27:55.714815",  "from": "tramp@q3mcco35auwcstmt.onion",  "to": "pumba@q3mcco35auwcstmt.onion",  "body": "saw"  }  {  "ts": "2022-01-09T11:28:23.092751",  "from": "pumba@q3mcco35auwcstmt.onion",  "to": "tramp@q3mcco35auwcstmt.onion",  "body": "so what do we do with them?"  } | [{  "ts": "2022-01-09T11:27:55.712408",  "from": "tramp@q3mcco35auwcstmt.onion",  "to": "pumba@q3mcco35auwcstmt.onion",  "body": "yes"  },{  "ts": "2022-01-09T11:27:55.714815",  "from": "tramp@q3mcco35auwcstmt.onion",  "to": "pumba@q3mcco35auwcstmt.onion",  "body": "saw"  },{  "ts": "2022-01-09T11:28:23.092751",  "from": "pumba@q3mcco35auwcstmt.onion",  "to": "tramp@q3mcco35auwcstmt.onion",  "body": "so what do we do with them?"  }] |

**f.write(fixed\_data)**

As the data is ready now, ‘json.load()’ function has been loaded into Colab Notebook. Pandas dataframe was used to combine the datasets. For each JSON file, we loaded the data into a dataframe, added identifier attributes for year and filename, and then appended. Finally, we combine all the dataframes into a single dataframe ‘df\_combined’

.

**Code:**

**import pandas as pd**

**import json**

**import os**

**def load\_json\_data(file\_path):**

**with open(file\_path, 'r') as f:**

**data = json.load(f)**

**return data**

**def load\_combine\_data(directory):**

**data\_list = []**

**for root, dirs, files in os.walk(directory):**

**for file in files:**

**file\_path = os.path.join(root, file)**

**year = os.path.basename(os.path.dirname(file\_path))**

**filename = os.path.basename(file\_path)**

**try:**

**json\_data = load\_json\_data(file\_path)**

**for item in json\_data:**

**item['year'] = year**

**item['filename'] = filename**

**data\_list.append(item)**

**except Exception as e:**

**print(f"Error loading file {file\_path}: {e}")**

**df = pd.DataFrame(data\_list)**

**return df**

**selected\_data\_dir = '/content/drive/My Drive/deepl\_translated\_jabber/preprocessed2'**

**df\_combined = load\_combine\_data(selected\_data\_dir)**

**print(df\_combined.head())**

**Output:**

**ts from \**

**0 2020-10-08T00:39:58.424869 target@q3mcco35auwcstmt.onion**

**1 2020-10-08T00:40:02.225972 target@q3mcco35auwcstmt.onion**

**2 2020-10-08T00:40:03.357210 target@q3mcco35auwcstmt.onion**

**3 2020-10-08T00:40:05.099278 target@q3mcco35auwcstmt.onion**

**4 2020-10-08T00:40:08.267454 target@q3mcco35auwcstmt.onion**

**to body year \**

**0 professor@q3mcco35auwcstmt.onion hahahahahahahaha 2020**

**1 professor@q3mcco35auwcstmt.onion from : office 2020**

**2 professor@q3mcco35auwcstmt.onion asking 2020**

**3 professor@q3mcco35auwcstmt.onion well. 2020**

**4 professor@q3mcco35auwcstmt.onion timelines you 2020**

**filename**

**0 185.25.51.173-20201008\_en-US.json**

**1 185.25.51.173-20201008\_en-US.json**

**2 185.25.51.173-20201008\_en-US.json**

**3 185.25.51.173-20201008\_en-US.json**

**4 185.25.51.173-20201008\_en-US.json**

**Data transformation**

**Task 2: Rename the identifiers (emails) into a simpler format (e.g. “ID001”), and keep the mapping from new name to old name as a dictionary.**

A list of unique email identifiers obtained by concatenating the 'from' and 'to' columns of the combined DataFrame and removing duplicates. For each email, we generate a new identifier in the format "ID001", "ID002", etc., using f-strings with zero-padding to ensure a consistent length of three digits. Finally, we print the email\_id\_mapping dictionary, which contains the mapping from old email identifiers to new ones.

**Code:**

email\_id\_mapping = {}

unique\_emails = pd.unique(df\_combined[['from', 'to']].values.ravel('K'))

for i, email in enumerate(unique\_emails, start=1):

email\_id\_mapping[email] = f"ID{i:03d}"

print(email\_id\_mapping)

**Output:**

**{'target@q3mcco35auwcstmt.onion': 'ID001', 'professor@q3mcco35auwcstmt.onion': 'ID002', 'zloysobaka@q3mcco35auwcstmt.onion': 'ID003',.....}**

**Task 3: Concatenate the text for each individual**

‘iterrows()’ method has been used to iterate through each row of the combined DataFrame. For each row, it retrieves the sender's email address ('from'), recipient's email address ('to'), and the body text of the message. It then maps the sender's email address to their corresponding simplified identifier using the email\_id\_mapping dictionary generated in Task 2.

The body text is concatenated for each individual based on their identifier. If the identifier already exists in the concatenated\_bodies dictionary, the body text is appended to the existing text. Otherwise, a new entry is created in the dictionary with the identifier as the key and the body text as the value.

**Code:**

**from tqdm import tqdm**

**concatenated\_bodies = {}**

**for index, row in tqdm(df\_combined.iterrows()):**

**email\_from = row['from']**

**email\_to = row['to']**

**body = row['body']**

**email\_id = email\_id\_mapping.get(email\_from)**

**if email\_id in concatenated\_bodies:**

**concatenated\_bodies[email\_id] = concatenated\_bodies[email\_id]+" "+body**

**else:**

**concatenated\_bodies[email\_id] = body**

**print(concatenated\_bodies)**

**Task 4: Using the “from” and “to” fields create a list of weighted edges or connections.**

A list of weighted edges constructed based on sender and recipient email addresses. These email addresses were mapped to simplified identifiers. For each unique edge (sender, recipient), a weight representing the number of occurrences of the connection was calculated. Finally, a list of weighted edges was generated, containing source, target, and weight information.

**Code:**

**weighted\_edges = {}**

**for index, row in df\_combined.iterrows():**

**email\_from = row['from']**

**email\_to = row['to']**

**edge = (email\_id\_mapping.get(email\_from), email\_id\_mapping.get(email\_to))**

**weighted\_edges[edge] = weighted\_edges.get(edge, 0) + 1**

**weighted\_edges\_list = [(source, target, weight) for (source, target), weight in weighted\_edges.items()]**

**print(weighted\_edges\_list)**

**Output:**

**[('ID001', 'ID002', 795), ('ID002', 'ID001', 226), ('ID001', 'ID004', 456), ('ID003', 'ID004', 6), ....]**

**Task 5: Include a list comprehension in one or more of the above solutions.**

For the solution 2 and 3, we used list comprehensions which makes the code more code more concise and elegant.

**Code: (list comprehension for solution 2)**

**email\_id\_mapping = {email: f"ID{i+1:03d}" for i, email in enumerate(pd.unique(df\_combined[['from', 'to']].values.ravel('K')))}**

**print(email\_id\_mapping)**

**Output:**

**{'target@q3mcco35auwcstmt.onion': 'ID001', 'professor@q3mcco35auwcstmt.onion': 'ID002', 'zloysobaka@q3mcco35auwcstmt.onion': 'ID003', 'stern@q3mcco35auwcstmt.onion': 'ID004',. . .}**

**Code: (list comprehension for solution 3)**

**from tqdm import tqdm**

**concatenated\_bodies = {email\_id\_mapping.get(row['from']): concatenated\_bodies.get(email\_id\_mapping.get(row['from']), '') + ' ' + row['body'] for \_, row in tqdm(df\_combined.iterrows())}**

**print(concatenated\_bodies)**

**Output:**

**23873it [00:01, 14624.31it/s]**

**{'ID001': "hahahahahahahaha from : office asking well. timelines you as they say fuck yeah fuck them from the turntable and . . .**

**Visualization and initial analysis**

For analysis and visualizations we used various data visualization techniques using the Python programming language (Srinivas, 2024).

**Task 6:Determine the date ranges ("ts" values) and frequency of activity for some of the members. Use GroupBy to achieve this.**

These Id: 'ID001', 'ID002', 'ID003', 'ID004', 'ID005', 'ID006', 'ID007', 'ID008', 'ID009', 'ID010' were selected for this task. Using pandas' GroupBy function, date ranges and activity frequency are calculated for selected members (Stack Overflow, 2015).

**Code:**

**df\_combined['ts'] = pd.to\_datetime(df\_combined['ts'])**

**member\_groups = df\_combined.groupby('from')**

**selected\_ids = ['ID001', 'ID002', 'ID003', 'ID004', 'ID005', 'ID006', 'ID007', 'ID008', 'ID009', 'ID010']**

**results = {}**

**for selected\_id in selected\_ids:**

**selected\_key = next((key for key, value in email\_id\_mapping.items() if value == selected\_id), None)**

**if selected\_key:**

**selected\_member\_activity = member\_groups.get\_group(selected\_key)**

**date\_range = selected\_member\_activity['ts'].min().date(), selected\_member\_activity['ts'].max().date()**

**activity\_frequency = len(selected\_member\_activity)**

**results[selected\_id] = {'date\_range': date\_range, 'activity\_frequency': activity\_frequency}**

**else:**

**results[selected\_id] = {'date\_range': 'N/A', 'activity\_frequency': 'N/A'}**

**for selected\_id, data in results.items():**

**print(f"Results for member {selected\_id}:")**

**print(f"Date range: {data['date\_range']}")**

**print(f"Frequency of activity: {data['activity\_frequency']}\n")**

**Output:**

**Results for member ID001:**

**Date range: (datetime.date(2020, 7, 9), datetime.date(2021, 10, 22))**

**Frequency of activity: 5305**

**Results for member ID002:**

**Date range: (datetime.date(2020, 7, 9), datetime.date(2021, 6, 28))**

**Frequency of activity: 462**

**Results for member ID003:**

**Date range: (datetime.date(2020, 10, 8), datetime.date(2020, 10, 19))**

**Frequency of activity: 26**

**Results for member ID004:**

**Date range: (datetime.date(2020, 7, 9), datetime.date(2022, 2, 21))**

**Frequency of activity: 1514**

**Results for member ID005:**

**Date range: (datetime.date(2020, 9, 18), datetime.date(2020, 10, 19))**

**Frequency of activity: 395**

**Results for member ID006:**

**Date range: (datetime.date(2020, 7, 23), datetime.date(2022, 1, 17))**

**Frequency of activity: 282**

**Results for member ID007:**

**Date range: (datetime.date(2020, 7, 9), datetime.date(2022, 2, 15))**

**Frequency of activity: 252**

**Results for member ID008:**

**Date range: (datetime.date(2020, 7, 23), datetime.date(2022, 1, 23))**

**Frequency of activity: 323**

**Results for member ID009:**

**Date range: (datetime.date(2020, 10, 8), datetime.date(2020, 10, 19))**

**Frequency of activity: 4**

**Results for member ID010:**

**Date range: (datetime.date(2020, 7, 9), datetime.date(2021, 11, 12))**

**Frequency of activity: 114**

**Task 7:Slice the data into months and use a Histogram to display the amount of activity per month for each of the years.**

Here ‘matplotlib.pyplot’ was used to slice the data into months and plotted a Histogram to visualize the count of activities per month for each year.

**Code:**

**import matplotlib.pyplot as plt**

**df\_combined['ts'] = pd.to\_datetime(df\_combined['ts'])**

**df\_combined['year'] = df\_combined['ts'].dt.year**

**df\_combined['month'] = df\_combined['ts'].dt.month**

**activity\_per\_month = df\_combined.groupby(['month','year' ]).size().unstack(fill\_value=0)**

**activity\_per\_month.plot(kind='bar', figsize=(10, 6))**

**plt.xlabel('Month')**

**plt.ylabel('Activity Count')**

**plt.title('Activity per Month for Each Year')**

**plt.legend(title='Year', bbox\_to\_anchor=(1, 1))**

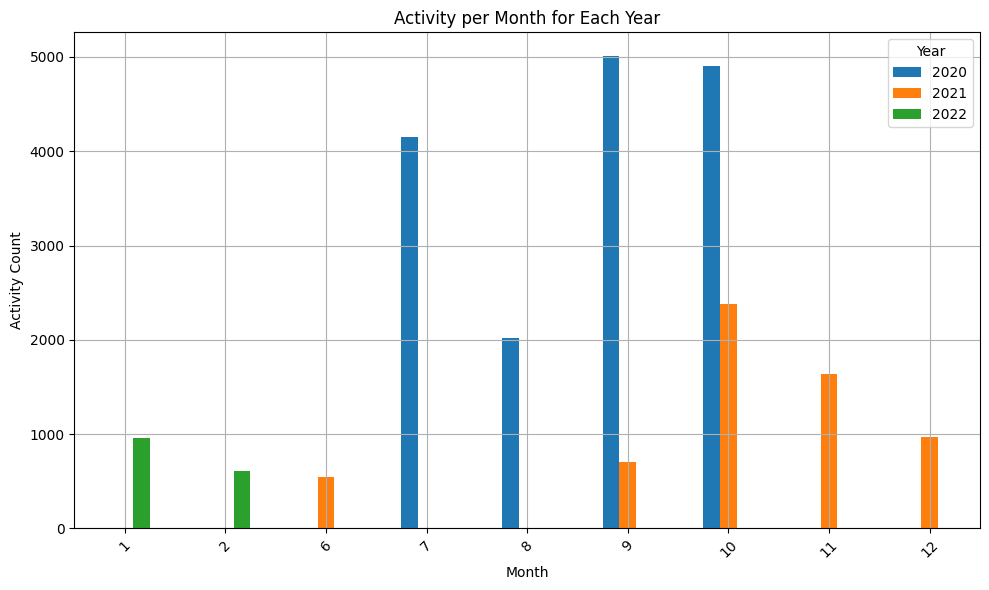
**plt.xticks(rotation=45)**

**plt.grid(True)**

**plt.tight\_layout()**

**plt.show()**

**print(activity\_per\_month)**

**Output:**

**Network analysis**

**Task 8:Use the python library networkx to display your data as a directed weighted graph.**

Here ‘weighted\_edges\_list’ has been used that was created before which contains source, target, and weight information. The ‘networkx’ library (Hagberg et al., 2008) used to create weighted graph. A directed graph (DiGraph) object is created using nx.DiGraph(). Nodes and edges are added to the graph using a for loop iterating over the weighted\_edges\_list. Node positions are determined using nx.shell\_layout(). The graph is then visualized using nx.draw() with specified parameters for node size, color, font size, and arrow size. Edge weights are displayed using nx.draw\_networkx\_edge\_labels(). Finally, the graph is displayed using plt.show().

And another one created using nx.random\_layout(G), which positions the nodes randomly within the plot.

**Code:**

**import pandas as pd**

**import networkx as nx**

**import matplotlib.pyplot as plt**

**G = nx.DiGraph()**

**for edge in weighted\_edges\_list:**

**from\_node, to\_node, weight = edge**

**G.add\_node(from\_node)**

**G.add\_node(to\_node)**

**for edge in weighted\_edges\_list:**

**from\_node, to\_node, weight = edge**

**G.add\_edge(from\_node, to\_node, weight=weight)**

**Code:(shell\_layout() method)**

**plt.figure(figsize=(50, 50))**

**pos = nx.shell\_layout(G)**

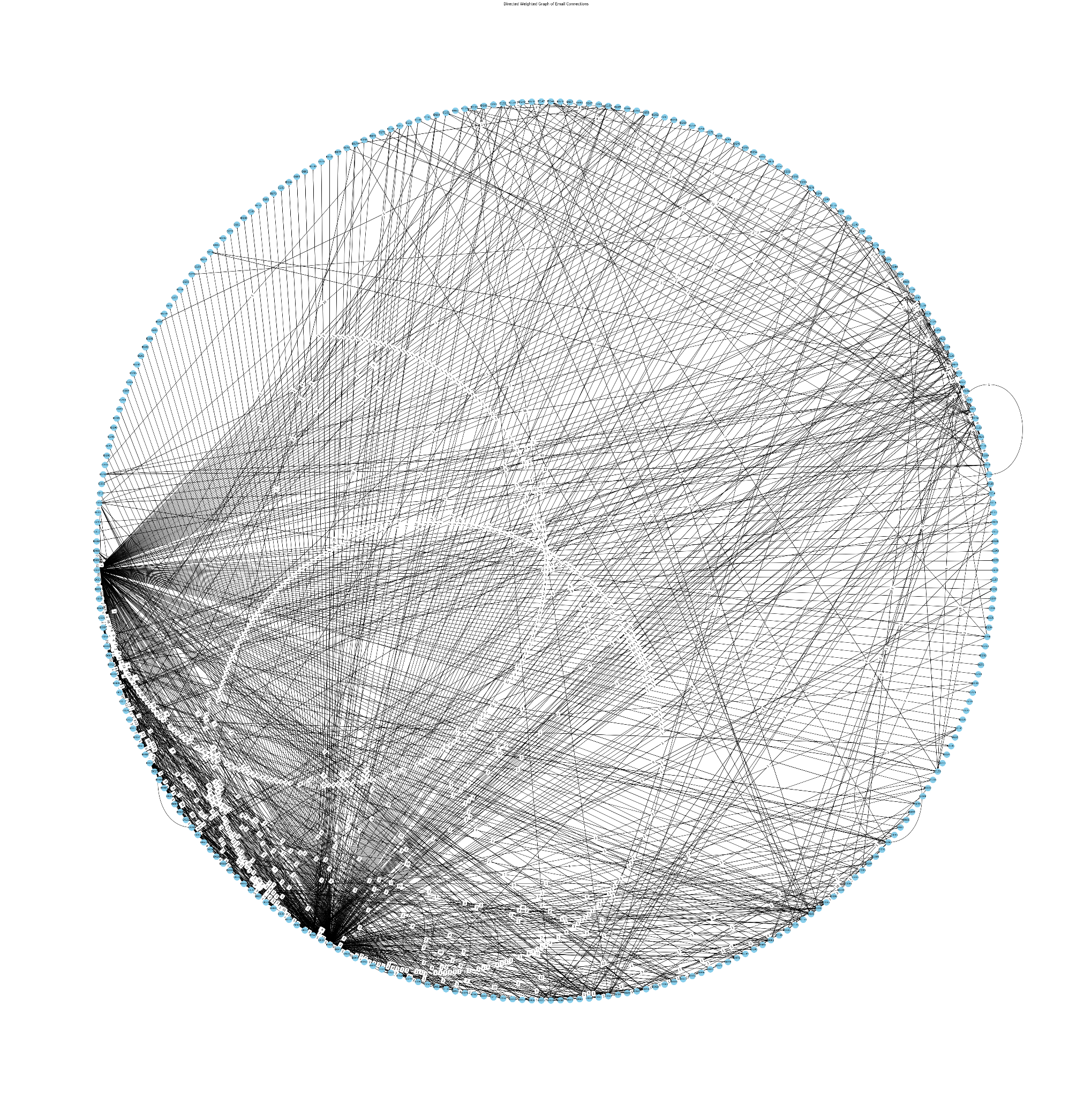
**nx.draw(G, pos, with\_labels=True, node\_size=400, node\_color="skyblue", font\_size=8, arrowsize=5)**

**labels = nx.get\_edge\_attributes(G, 'weight')**

**nx.draw\_networkx\_edge\_labels(G, pos, edge\_labels=labels)**

**plt.title("Directed Weighted Graph of Email Connections")**

**plt.show()**

**Output:**

**Code:(random\_layout() method)**

**plt.figure(figsize=(50, 50))**

**pos = nx.random\_layout(G)**

**nx.draw(G, pos, with\_labels=True, node\_size=400, node\_color="skyblue", font\_size=8, arrowsize=5)**

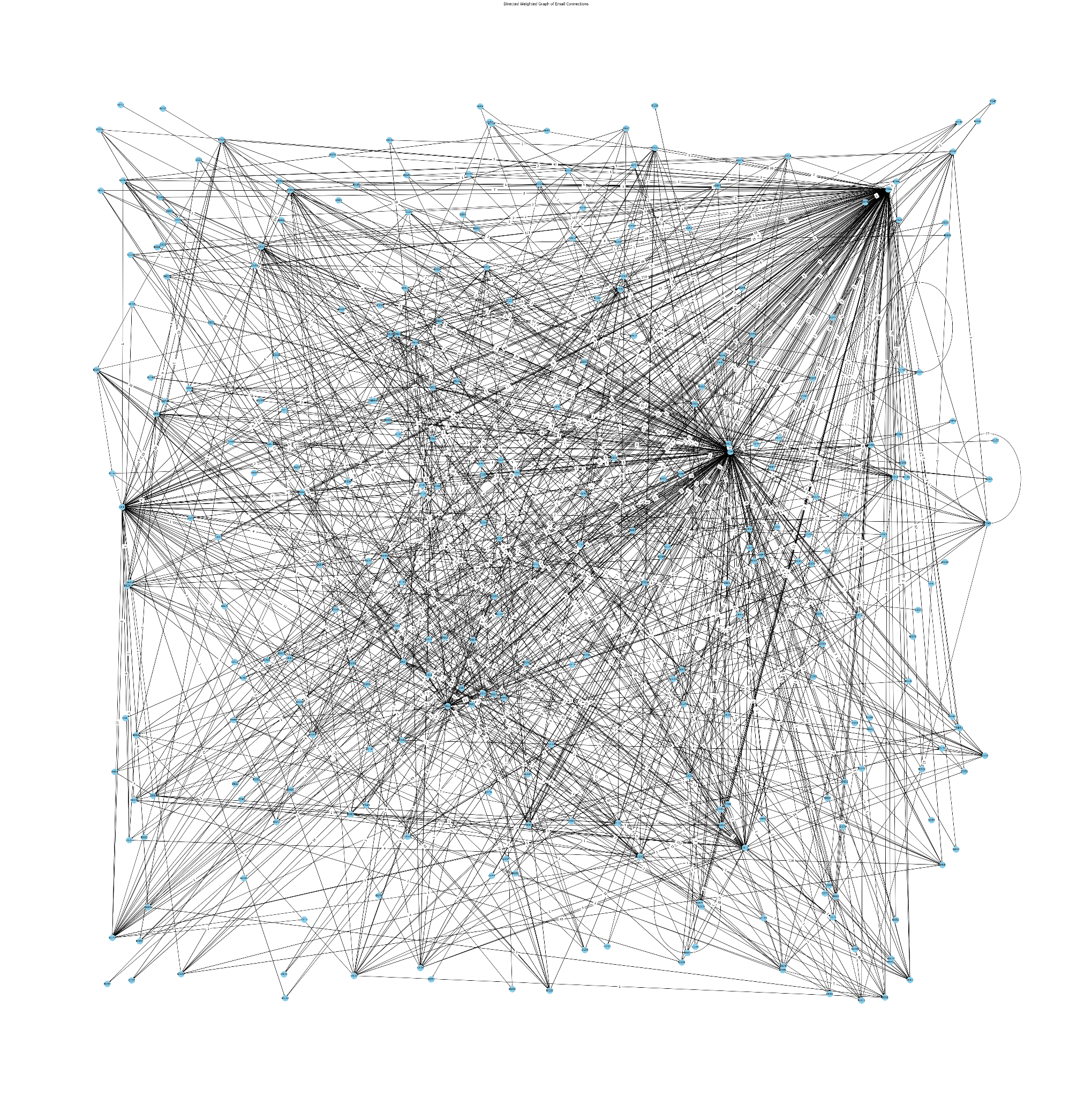
**labels = nx.get\_edge\_attributes(G, 'weight')**

**nx.draw\_networkx\_edge\_labels(G, pos, edge\_labels=labels)**

**plt.title("Directed Weighted Graph of Email Connections")**

**plt.show()**

**Output:**

****

**Task 9:Calculate the centrality measures of Degree and Betweenness for each of the nodes (members) in the gang.**

The degree centrality measure is calculated for each node in the graph using nx.degree\_centrality(). A bar plot is created to visualize the degree centrality measures for each node. The x-axis represents the nodes, and the y-axis represents the degree centrality.

The betweenness centrality measure is calculated for each edge in the graph using nx.betweenness\_centrality(). The normalized parameter is set to True to normalize the betweenness values. A bar plot is created to visualize the betweenness centrality measures for each edge.

**Code: (Degree Centrality)**

**import networkx as nx**

**import matplotlib.pyplot as plt**

**simple\_degree\_centrality = nx.degree\_centrality(G)**

**plt.figure(figsize=(40, 6))**

**plt.bar(range(len(G.nodes())), list(simple\_degree\_centrality.values()), color='blue', label='Simple Degree Centrality')**

**plt.xlabel('Nodes')**

**plt.ylabel('Degree Centrality')**

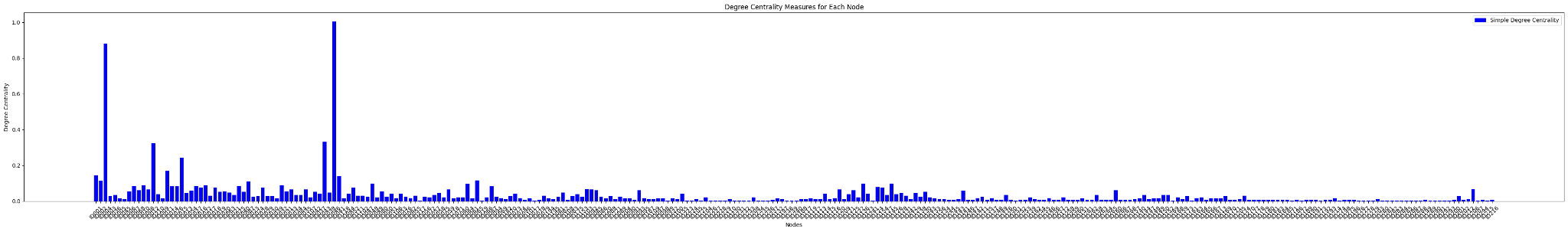
**plt.title('Degree Centrality Measures for Each Node')**

**plt.xticks(range(len(G.nodes())), G.nodes(), rotation=45)**

**plt.legend()**

**plt.tight\_layout()**

**plt.show()**

**Output:**

**Code: (Betweenness Centrality)**

**betweenness\_centrality = nx.betweenness\_centrality(G, normalized = True, endpoints = False)**

**edges = list(G.edges())**

**betweenness\_values = [betweenness\_centrality[edge[0]] + betweenness\_centrality[edge[1]] for edge in edges]**

**plt.figure(figsize=(100, 6))**

**plt.bar(range(len(edges)), betweenness\_values, color='red', label='Betweenness Centrality', alpha=0.5)**

**plt.xlabel('Edges')**

**plt.ylabel('Centrality Measures')**

**plt.title('Centrality Measures for Each Edge')**

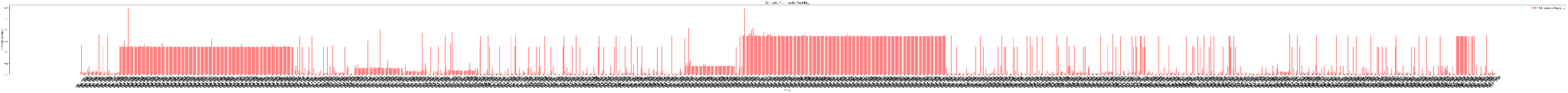
**plt.xticks(range(len(edges)), [f"{edge[0]} - {edge[1]}" for edge in edges], rotation=45)**

**plt.legend()**

**plt.tight\_layout()**

**plt.show()**

**Output:**

****

**Task 10:Who seem to be the most important members and why?**

To determine the most important members in the network, we can consider both degree centrality and betweenness centrality.

1. Degree Centrality: Nodes with high degree centrality have many connections(edges) in the network. They are important because they directly interact with many other members.
2. Betweenness Centrality: Nodes with high betweenness centrality lie on many shortest paths between other nodes in the network. They are important because they act as intermediaries facilitating communication between different parts of the network.

By considering both metrics, we can identify nodes that are not only highly connected but also play a critical role in facilitating communication between other nodes.

**Code:**

**top\_degree\_members = sorted(simple\_degree\_centrality.items(), key=lambda x: x[1], reverse=True)[:5]**

**top\_betweenness\_members = sorted(betweenness\_centrality.items(), key=lambda x: x[1], reverse=True)[:5]**

**print("Top 5 members based on degree centrality:")**

**for member, centrality in top\_degree\_members:**

**print(f"Member: {member}, Degree Centrality: {centrality}")**

**print("\nTop 5 members based on betweenness centrality:")**

**for member, centrality in top\_betweenness\_members:**

**print(f"Member: {member}, Betweenness Centrality: {centrality}")**

**Output:**

**Top 5 members based on degree centrality:**

**Member: ID046, Degree Centrality: 1.0034129692832765**

**Member: ID004, Degree Centrality: 0.8805460750853242**

**Member: ID043, Degree Centrality: 0.3310580204778157**

**Member: ID012, Degree Centrality: 0.3242320819112628**

**Member: ID015, Degree Centrality: 0.24232081911262798**

**Top 5 members based on betweenness centrality:**

**Member: ID046, Betweenness Centrality: 0.34672987500105784**

**Member: ID004, Betweenness Centrality: 0.2495359436577064**

**Member: ID043, Betweenness Centrality: 0.07323412790158498**

**Member: ID012, Betweenness Centrality: 0.05684500293538243**

**Member: ID015, Betweenness Centrality: 0.03164994867250837**

# **Natural Language Processing**

To clean the chats, we used Natural Language Processing (NLP). NLP is a subfield of artificial intelligence that focuses on allowing a machine to understand natural language, that is human language (Chowdhary, 2020),(Raina,2022). Basically, NLP teaches a machine to learn, understand and derive meaning from a language. Natural language processing uses various algorithms to learn and follow grammatical rules which are then used to derive meaning out of words and sentences (Chowdhary, 2020),(Raina,2022). Some of the most commonly used algorithms are stemming (reducing words to their lexical root), lemmatization (converting a word into its canonical form) and tokenization ( Ruellan,2023).

**(Sentiment analysis)**

**Task 11: Use the Vader library for sentiment analysis on the concatenated text you prepared earlier. Store the results for each person.**

‘SentimentIntensityAnalyzer’ is used from Vader library to perform sentiment analysis on the concatenated text prepared earlier (Hutto, 2014). For each person, the polarity\_scores() method of the analyzer object is called to obtain sentiment scores, including negative (neg), neutral (neu), positive (pos), and compound (compound) scores. The sentiment scores for each person are stored in the sentiment\_scores dictionary. Finally, a DataFrame is created from the sentiment\_scores dictionary, where each row represents a person, and columns represent sentiment scores (neg, neu, pos, compound).

**Code:**

**from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer**

**import pandas as pd**

**analyzer = SentimentIntensityAnalyzer()**

**sentiment\_scores = {}**

**for person, text in concatenated\_bodies.items():**

**sentiment\_scores[person] = analyzer.polarity\_scores(text)**

**sentiment\_df = pd.DataFrame(sentiment\_scores).T**

**sentiment\_df.index.name = 'Person'**

**print("Sentiment scores for each person:")**

**print(sentiment\_df)**

**Output:**

Sentiment scores for each person:

neg neu pos compound

Person

ID001 0.097 0.778 0.125 1.0000

ID002 0.081 0.834 0.085 -0.9718

ID003 0.043 0.876 0.081 0.7941

ID004 0.024 0.868 0.108 1.0000

ID005 0.025 0.830 0.144 0.9994

... ... ... ... ...

ID212 0.000 1.000 0.000 0.0000

ID213 0.000 0.770 0.230 0.7170

ID214 0.000 1.000 0.000 0.0000

ID215 0.000 1.000 0.000 0.0000

ID216 0.000 1.000 0.000 0.0000

[216 rows x 4 columns]

**(Technical lexicon)**

**Task 12: A lot of the messages between members contain technical terms related to ransomware scripts. Construct a simple lexicon with 10 terms and use this to analyse your text data.**

Lexicon has been Constructed with a simple lexicon containing 10 terms related to ransomware scripts to analyze the text data. These terms are:domain, server, encryption, crypt, malicious, code, crypto, files, data, deploy. For each member, the occurrences of each term in their messages has been counted.

**Code:**

**import re**

**lexicon = ['domain', 'server', 'encryption', 'crypt', 'malicious', 'code', 'crypto', 'files', 'data', 'deploy']**

**term\_frequencies = {person: {term: 0 for term in lexicon} for person in concatenated\_bodies}**

**for person, text in concatenated\_bodies.items():**

**text\_lower = text.lower()**

**for term in lexicon:**

**term\_occurrences = re.findall(r'\b' + re.escape(term) + r'\b', text\_lower)**

**term\_frequencies[person][term] = len(term\_occurrences)**

**for person, frequencies in term\_frequencies.items():**

**print(f"Term frequencies for {person}:")**

**for term, frequency in frequencies.items():**

**print(f"{term}: {frequency}")**

**print()**

**Output:**

**Term frequencies for ID001:**

**domain: 25**

**server: 22**

**encryption: 0**

**crypt: 9**

**malicious: 0**

**code: 6**

**crypto: 14**

**files: 14**

**data: 10**

**deploy: 3**

**Term frequencies for ID002:**

**domain: 9**

**server: 2**

**encryption: 0**

**crypt: 2**

**malicious: 0**

**code: 1**

**crypto: 1**

**files: 4**

**data: 8**

**deploy: 1**

**Term frequencies for ID003:**

**domain: 0**

**server: 0**

**encryption: 0**

**crypt: 0**

**malicious: 0**

**code: 0**

**crypto: 1**

**files: 0**

**data: 0 ...**

**Code: (plot graph to analyse ‘Word counts for each person’)**

**import matplotlib.pyplot as plt**

**x\_values = list(concatenated\_bodies.keys())**

**legend\_labels = lexicon**

**legend\_colors = plt.cm.get\_cmap('tab10', len(lexicon))**

**bottom = [0] \* len(x\_values)**

**plt.figure(figsize=(50, 6))**

**for i, term in enumerate(lexicon):**

**y\_values = [term\_frequencies[person][term] for person in concatenated\_bodies]**

**plt.bar(x\_values, y\_values, bottom=bottom, label=term, color=legend\_colors(i))**

**bottom = [bottom[j] + y\_values[j] for j in range(len(x\_values))]**

**plt.xlabel('Person ID')**

**plt.ylabel('Word Count')**

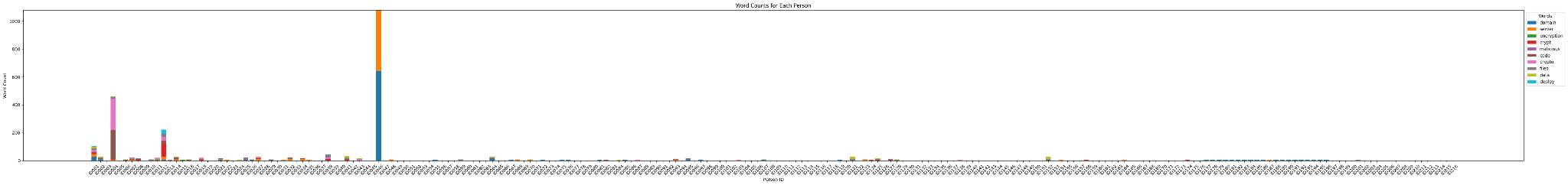
**plt.title('Word Counts for Each Person')**

**plt.legend(loc='upper left', bbox\_to\_anchor=(1, 1), title='Words')**

**plt.xticks(rotation=45)**

**plt.tight\_layout()**

**plt.show()**

****

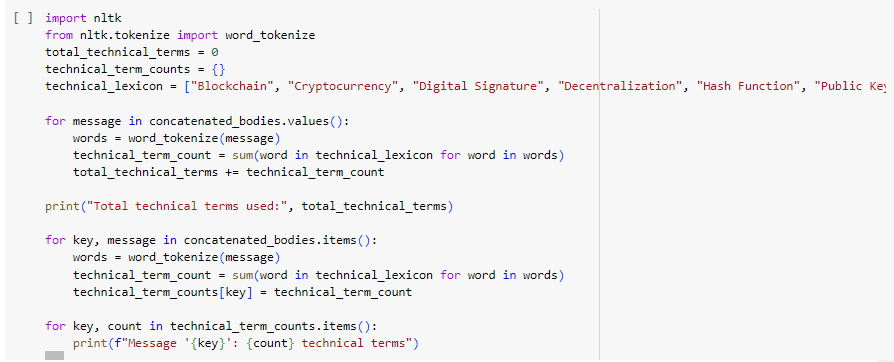
**Task 13: Analyse the text data.**

**1. How many contain technical terms? How many technical terms are used?**

A list named technical\_lexicon is defined, containing a comprehensive set of technical terms related to various domains such as blockchain, cybersecurity, cryptocurrency, malware, networking, etc. These terms will be used to identify technical terms within the text data. Total 2924 technical terms used in the concatenated\_bodies.

And Number of messages with technical terms is 105.

**Code:**

****

**Output:**

**Total technical terms used: 2924**

**Message 'ID001': 93 technical terms**

**Message 'ID002': 29 technical terms**

**Message 'ID003': 4 technical terms**

**Message 'ID004': 669 technical terms**

**Message 'ID005': 0 technical terms**

**Message 'ID006': 12 technical terms**

**Message 'ID007': 26 technical terms**

**Message 'ID008': 17 technical terms ...**

**Number of messages containing technical terms: 109**

**2. How would you find out the complete list of terms in your data?**

I can iterate through each message, tokenize the words and compile a set of unique terms.

**Code:**

**unique\_technical\_terms = set()**

**for message in concatenated\_bodies.values():**

**words = word\_tokenize(message)**

**for word in words:**

**if word in technical\_lexicon:**

**unique\_technical\_terms.add(word)**

**print("Complete list of terms in the data:")**

**print(unique\_technical\_terms)**

**Output:**

**Complete list of terms in the data:**

**{'Server', 'crypto', 'Virtualization', 'Network', 'Linux', 'HTTP', 'Git', 'NFT', 'Debugging', 'deploy', 'Policy', 'Platform', 'VM', 'Database', 'Ransomware', 'files', 'Program', 'VPN', 'code', 'Backdoor', 'domain', 'Endpoint', 'RAM', 'XML', 'Java', 'Function', 'encryption', 'crypt', 'Service', 'data', 'API', 'Tool', 'Ethereum', 'SQL', 'server', 'URL'}**

**3. WHO is telling WHO about technical matters.**

To analyze the data to find out who is messaging whom on technical matters. We can use df.combined to iterate through each message to check if it contains technical terms, and then extract the sender and recipient information.

**Code:**

**technical\_discussions = {}**

**for index, row in df\_combined.iterrows():**

**message = row['body']**

**sender = row['from']**

**recipient = row['to']**

**if any(term in message.lower() for term in technical\_lexicon):**

**if sender in technical\_discussions:**

**if recipient in technical\_discussions[sender]:**

**technical\_discussions[sender][recipient] += 1**

**else:**

**technical\_discussions[sender][recipient] = 1**

**else:**

**technical\_discussions[sender] = {recipient: 1}**

**for sender, recipients in technical\_discussions.items():**

**for recipient, count in recipients.items():**

**print(f"{sender} is talking technical matters with {recipient} ({count} messages)")**

**Output:**

**target@q3mcco35auwcstmt.onion is talking technical matters with professor@q3mcco35auwcstmt.onion (9 messages)**

**target@q3mcco35auwcstmt.onion is talking technical matters with troy@q3mcco35auwcstmt.onion (24 messages)**

**target@q3mcco35auwcstmt.onion is talking technical matters with voron@q3mcco35auwcstmt.onion (12 messages)**

**target@q3mcco35auwcstmt.onion is talking technical matters with stern@q3mcco35auwcstmt.onion (17 messages)**

**target@q3mcco35auwcstmt.onion is talking technical matters with reshaev@q3mcco35auwcstmt.onion (1 messages) ...**

**4. What kinds of messages are there? How many messages are just random chitchat?**

To find out what kind of messages are there, we made some lexicons based on discussion categories.Then we iterate through df.combined and check if the messages belong to any lexicons or not. The result shows that the number of chitchat messages are 3636, on the other hand technical messages are 3152. Given that the data is from a private group focused on ransomware, one might expect a higher frequency of technical terms compared to casual conversation. Besides , 10501 messages belong to the slang category and 5609 messages can't be identified.

**Code:**

****

**for index, row in df\_combined.iterrows():**

**message = row['body'].lower()**

**if any(term in message for term in technical\_lexicon):**

**category\_counts["technical"] += 1**

**elif any(term in message for term in buy\_selling\_lexicon):**

**category\_counts["buy/selling"] += 1**

**elif any(term in message for term in movie\_discussion\_lexicon):**

**category\_counts["movie discussion"] += 1**

**elif any(term in message for term in sports\_lexicon):**

**category\_counts["sports"] += 1**

**elif any(term in message for term in chitchat\_lexicon):**

**category\_counts["chitchats"] += 1**

**elif any(term in message for term in politics\_lexicon):**

**category\_counts["politics"] += 1**

**elif any(term in message for term in music\_lexicon):**

**category\_counts["music"] += 1**

**elif any(term in message for term in food\_lexicon):**

**category\_counts["food"] += 1**

**elif any(term in message for term in technology\_lexicon):**

**category\_counts["technology"] += 1**

**elif any(term in message for term in travel\_lexicon):**

**category\_counts["travel"] += 1**

**elif any(term in message for term in books\_lexicon):**

**category\_counts["books"] += 1**

**elif any(term in message for term in current\_affairs\_lexicon):**

**category\_counts["current affairs"] += 1**

**elif any(term in message for term in personal\_development\_lexicon):**

**category\_counts["personal development"] += 1**

**elif any(term in message for term in slangs\_lexicon):**

**category\_counts["slangs"] += 1**

**else:**

**category\_counts["other"] += 1**

**for category, count in category\_counts.items():**

**print("Number of messages in", category, ":", count)**

**Output:**

**Number of messages in technical : 3152**

**Number of messages in buy/selling : 213**

**Number of messages in movie discussion : 36**

**Number of messages in sports : 223**

**Number of messages in chitchats : 3636**

**Number of messages in politics : 19**

**Number of messages in music : 34**

**Number of messages in food : 41**

**Number of messages in technology : 196**

**Number of messages in travel : 37**

**Number of messages in books : 41**

**Number of messages in current affairs : 113**

**Number of messages in personal development : 22**

**Number of messages in slangs : 10501**

**Number of messages in other : 5609**

**5. Are there any power structures visible through analysis of the language (‘you need to do this’, etc).**

To analyze the data to find out who is messaging whom on technical matters. We can use df.combined to iterate through each message to check if it contains technical terms, and then extract the sender and recipient information.

**Code:**

**import re**

**authority\_patterns = [**

**r"\b(?:you|u)\s\*must\b",**

**r"\b(?:you|u)\s\*need\s\*to\b",**

**r"\b(?:you|u)\s\*have\s\*to\b",**

**r"\b(?:you|u)\s\*should\b",**

**r"\b(?:you|u)\s\*shall\b",**

**r"\b(?:you|u)\s\*are\s\*required\s\*to\b",**

**r"\b(?:you|u)\s\*are\s\*obligated\s\*to\b",**

**r"\b(?:you|u)\s\*are\s\*expected\s\*to\b",**

**r"\b(?:you|u)\s\*are\s\*commanded\s\*to\b",**

**r"\b(?:you|u)\s\*are\s\*advised\s\*to\b",**

**r"\b(?:you|u)\s\*are\s\*urged\s\*to\b",**

**r"\b(?:you|u)\s\*are\s\*encouraged\s\*to\b",**

**r"\b(?:you|u)\s\*are\s\*compelled\s\*to\b",**

**r"\b(?:you|u)\s\*are\s\*supposed\s\*to\b",**

**r"\b(?:you|u)\s\*are\s\*instructed\s\*to\b",**

**r"\b(?:you|u)\s\*are\s\*told\s\*to\b",**

**r"\b(?:you|u)\s\*are\s\*commanded\s\*to\b",**

**r"\b(?:you|u)\s\*are\s\*ordered\s\*to\b",**

**r"\b(?:you|u)\s\*are\s\*dictated\s\*to\b",**

**r"\b(?:you|u)\s\*are\s\*directed\s\*to\b",**

**r"\b(?:you|u)\s\*are\s\*supervised\s\*to\b",**

**r"\b(?:you|u)\s\*are\s\*guided\s\*to\b",**

**r"\b(?:you|u)\s\*are\s\*prescribed\s\*to\b",**

**r"\b(?:you|u)\s\*are\s\*expected\s\*to\b",**

**]**

**sender\_authority\_counts = {}**

**for index, row in df\_combined.iterrows():**

**message = row['body'].lower()**

**sender = row['from']**

**authority\_count = sum(1 for pattern in authority\_patterns if re.search(pattern, message))**

**if sender not in sender\_authority\_counts:**

**sender\_authority\_counts[sender] = authority\_count**

**else:**

**sender\_authority\_counts[sender] += authority\_count**

**sorted\_sender\_authority\_counts = dict(sorted(sender\_authority\_counts.items(), key=lambda item: item[1], reverse=True))**

**for sender, count in sorted\_sender\_authority\_counts.items():**

**print("Sender:", sender, "-->Power Pattern Count:", count)**

**Output:  
Sender: ttrr@conference.q3mcco35auwcstmt.onion -->Power Pattern Count: 24**

**Sender: bentley@q3mcco35auwcstmt.onion -->Power Pattern Count: 13**

**Sender: mango@q3mcco35auwcstmt.onion -->Power Pattern Count: 8**

**Sender: target@q3mcco35auwcstmt.onion -->Power Pattern Count: 7**

**Sender: hof@q3mcco35auwcstmt.onion -->Power Pattern Count: 6**

**Sender: pumba@q3mcco35auwcstmt.onion -->Power Pattern Count: 6**

**Sender: professor@q3mcco35auwcstmt.onion -->Power Pattern Count: 4**

**Sender: ghost@q3mcco35auwcstmt.onion -->Power Pattern Count: 4**

**Sender: viper@q3mcco35auwcstmt.onion -->Power Pattern Count: 4**

**Sender: bio@q3mcco35auwcstmt.onion -->Power Pattern Count: 4**

**Sender: bloodrush@q3mcco35auwcstmt.onion -->Power Pattern Count: 4**

**Sender: stern@q3mcco35auwcstmt.onion -->Po wer Pattern Count: 3**

**Sender: green@q3mcco35auwcstmt.onion -->Power Pattern Count: 3**

**Sender: price@q3mcco35auwcstmt.onion -->Power Pattern Count: 3**

**Sender: kaktus@q3mcco35auwcstmt.onion -->Power Pattern Count: 3**

**Sender: tramp@q3mcco35auwcstmt.onion -->Power Pattern Count: 3**

**6. What use is sentiment analysis for understanding this data.**

The sentiment scores (Kostadinov, 2022) provide insights into the emotional tone and attitude of each person's communication. Here's the impact of each sentiment score:

* **neg (Negative Sentiment):** Indicates the proportion of negative words in the text. Higher values suggest a more negative tone or expression of emotions in communication. For example: Person ID002 has a relatively high negative sentiment score (0.081), indicating a higher proportion of negative words in their communication.
* **neu (Neutral Sentiment):** Represents the proportion of neutral words in the text. Higher values suggest a more neutral tone without strong positive or negative emotions. For example: Person ID213 has a neutral sentiment score of 0.770, indicating a significant portion of their communication is neutral.
* **pos (Positive Sentiment):** Indicates the proportion of positive words in the text. Higher values suggest a more positive tone or expression of emotions in the communication. For example: Person ID005 has a relatively high positive sentiment score (0.144), indicating a higher proportion of positive words in their communication.
* **compound (Compound Sentiment):** Represents the overall sentiment of the text, combining the scores of neg, neu, and pos into a single metric. It ranges from -1 (extremely negative) to 1 (extremely positive). Scores closer to 0 suggest a more neutral sentiment. For example: Person ID002 has a low compound sentiment score (-0.9718), indicating a predominantly negative tone in their communication.

**What use is sentiment analysis for understanding this data?  
=>**

* **Identifying Communication Patterns:** Analyzing the sentiment of messages exchanged between individuals or groups can reveal patterns in communication dynamics.
* **Detecting Anomalies or Unusual Behavior:** Sudden changes in sentiment patterns, such as a spike in negative sentiment from a particular member, could signal potential problems such as harassment, bullying, or discontentment.

**Future work**

**Task 14: Discuss what experiments you would do next to understand the data better.**

1. **Cyber Activity Monitoring:** Monitor online activity based on the insights derived from the data analysis. By identifying technical discussions, sentiment trends, and communication patterns related to cybersecurity and ransomware, we can establish a framework for ongoing monitoring of cyber activity within different communities. This can involve tracking changes in technical vocabulary, detecting anomalous communication patterns that may indicate potential security threats, and staying vigilant for emerging trends or shifts in behavior that could signal cyberattacks or malicious activity. By leveraging the data-driven insights, we can proactively identify and respond to cybersecurity risks, enhance threat detection capabilities, and safeguard against potential security breaches.
2. **Network Analysis:** Construct a communication network graph where nodes represent members and edges represent interactions (e.g., messages exchanged). Analyze network centrality measures such as degree centrality, betweenness centrality, and eigenvector centrality to identify influential members and communication patterns within the group.
3. **Entity Recognition:** Use Named Entity Recognition (NER) to extract entities such as organizations, locations, and people mentioned in the messages. This can provide insights into the key entities of interest to the group and their relationships.
4. **Deep Learning Models:** Experiment with more advanced deep learning models for sentiment analysis, to capture more nuanced sentiment expressions and improve sentiment classification accuracy.

# **Theory Questions**

**Task 15: Describe the components of the data analytics process and illustrate your answer from the work you have carried out on the Conti-leaks data**

The data analytics process comprises several key components, each playing a crucial role in deriving insights and making informed decisions from data. These components include:

1. **Data Collection:** The initial step involves gathering raw data from various sources such as databases, files, APIs, or sensors. The data collected should be relevant to the objectives of the analysis and may include structured, semi-structured, or unstructured data.
2. **Data Preprocessing:** After the data collection, the raw data often requires preprocessing to clean, transform, and organize it for analysis. This involves tasks such as handling missing values, removing duplicates, standardizing formats, and encoding categorical variables. Data preprocessing ensures that the data is of high quality and ready for analysis.
3. **Exploratory Data Analysis (EDA):** EDA is an essential phase where analysts explore the characteristics of the data to gain insights and identify patterns or trends. Techniques such as summary statistics, data visualization, and correlation analysis are used to understand the distribution, relationships, and outliers within the data.
4. **Feature Engineering:** Feature engineering involves creating new features or transforming existing ones to improve the performance of machine learning models. This may include extracting relevant information from raw data, scaling features, encoding categorical variables, or creating interaction terms.
5. **Modeling:** In this component, statistical or machine learning models are applied to the prepared data to extract meaningful insights, make predictions, or solve specific problems. Models may range from simple regression or classification algorithms to complex deep learning architectures, depending on the nature of the data and the objectives of the analysis.
6. **Evaluation:** Once models are trained, they need to be evaluated to assess their performance and generalization ability. Evaluation metrics vary depending on the type of analysis and may include accuracy, precision, recall, F1-score, or area under the curve (AUC), among others.

**Illustration from the project:**

* **Data Collection:** Raw data in the form of JSON files containing leaked messages from Conti's XMPP chat server was collected. Which were released in Russian as 393 json files [5]
* **Data Preprocessing:** The raw data was cleaned by removing irrelevant metadata, restructuring the invalid format of data was performed and tokenizing message bodies to prepare it for analysis].
* **Exploratory Data Analysis:** EDA was conducted to visualize message frequency, identify technical terms related to ransomware, visualize the connection between them using graph plot, and detect communication patterns among members.
* **Feature Engineering:** New features such as message length, presence of technical terms, and sentiment scores were engineered to enhance the analysis of the data.
* **Modeling:** Sentiment analysis models were trained to analyze the emotional tone of messages and detect sentiment patterns among members.
* **Evaluation:** Evaluation metrics such as sentiment scores, communication patterns, and the presence of technical terms were used to assess the effectiveness of the analysis.

**Task 16: Where could you have used One-hot encoding in your experimentation?**

In this experimentation, one-hot encoding could be utilized to categorize the textual data based on different attributes or characteristics. For example, the textual data could be categorized based on various attributes such as the topic or subject matter of the messages, including categories like "Financial", "Legal", "Technical", "Operational", etc., depending on the content. Each category would be assigned a unique index, and then each message would be represented by a binary vector indicating its category membership. With the textual data categorized and represented using one-hot encoding, this structured data could be used to train a sentiment analysis model or any other classification model of interest.

**Task 17: Explain the terms sampling frame, population, and random sample, with respect to your work, and how this will have affected your results.**

For the sentiment analysis project on the Conti leaks data, key terms such as sampling frame, population, and random sample are explained below.

1. **Sampling Frame:**

The sampling frame refers to the collection of data or documents available for analysis. In my case, it encompasses the leaked JSON files containing messages from Conti's XMPP chat server. It represents the pool of data from which I select samples for sentiment analysis. In this case, I will separately choose 10 files from each year, so my sampling frame consists of the data from three years. The sampling frame includes all available messages from the specified timeframe, starting from January 21, 2020, up to the present date.

1. **Population:**

The population comprises the entire set of messages leaked from Conti's XMPP chat server during the specified period. This includes all 60,694 messages in 394 JSON files from January 2020 to February 2022. The population represents the full scope of data that I'm interested in analyzing for sentiment.

1. **Random Sample:**

A random sample is a subset of the population selected in such a way that every message has an equal chance of being included. In my project, I've randomly chosen 10 JSON files from each year to conduct sentiment analysis. This random selection approach ensures that each file has an equal likelihood of being included in the sample, reducing the risk of bias.

**Impact on Results:**

The sampling frame influences the representativeness of the data analyzed. If the sampling frame is limited or biased, it may not accurately reflect the diversity of messages within the population. One noteworthy point is that the size of different JSON files differs from each other, suggesting that files with lower content may have some missing context, which could influence the sentiment analysis (Alzahrani, 2022). Also, the population represents the entire dataset of leaked messages, and the accuracy of my results depends on how well my sample represents this population. Utilizing a random sample helps mitigate bias and increases the likelihood of generalizability. However, the size of the sample relative to the population and the representativeness of the sampling frame are crucial factors that could have affected the reliability and validity of my sentiment analysis results.

**References:**

1. Chowdhary, K. R. (2020). Natural language processing. In Springer eBooks (pp. 603–649). https://doi.org/10.1007/978-81-322-3972-7\_19
2. Raina, V., & Krishnamurthy, S. (2022). Building an effective data science practice. In Apress eBooks. https://doi.org/10.1007/978-1-4842-7419-4
3. Ruellan, E., Paquet-Clouston, M., & García, S. R. (2023). Conti Inc.: Understanding the Internal Discussions of a large Ransomware-as-a-Service Operator with Machine Learning. arXiv (Cornell University). https://doi.org/10.48550/arxiv.2308.16061
4. Alzahrani, S., Xiao, Y., & Sun, W. (2022). An Analysis of Conti Ransomware Leaked Source Codes. IEEE Access, 10, 100178–100193. https://doi.org/10.1109/access.2022.3207757
5. Kostadinov, B., Liu, J., & Rayme, J. (2022). Using Data Science Tools for Investigating Chat Logs from the Conti Ransomware Group. In 2022 IEEE 13th Annual Ubiquitous Computing, Electronics &amp; Mobile Communication Conference (UEMCON). 2022 IEEE 13th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON). IEEE. <https://doi.org/10.1109/uemcon54665.2022.9965691>
6. Srinivas, T. A. S., & Bharathi, M. (2024). Seeing Beyond Numbers: Mastering Data Visualization Interpretation. Zenodo.
7. Hagberg, Aric & Swart, Pieter & Chult, Daniel. (2008). Exploring Network Structure, Dynamics, and Function Using NetworkX. Proceedings of the 7th Python in Science Conference.
8. Stack Overflow. (2013, September 27). Read file into list and strip newlines. Retrieved from<https://stackoverflow.com/questions/19062574/read-file-into-list-and-strip-newlines>
9. Stack Overflow. (2013, August 29). How do I automatically fix an invalid JSON string? Retrieved from<https://stackoverflow.com/questions/18514910/how-do-i-automatically-fix-an-invalid-json-string>
10. Python Software Foundation. (n.d.). JSON encoder and decoder. Retrieved from<https://docs.python.org/3/library/json.html>
11. ContiLeaks. (2022). A cache of chat logs from the Conti group, in Russian. Retrieved from<https://share.vx-underground.org/Conti/>
12. Stack Overflow. (2015, August 20). Pandas GroupBy concatenates strings in multiple columns. Retrieved from<https://stackoverflow.com/questions/32117848/pandas-groupby-concatenate-strings-in-multiple-columns>
13. Hutto, C. J. (2014). VaderSentiment. GitHub. Retrieved from<https://github.com/cjhutto/vaderSentiment>