Load Dataset

```
import pandas as pd
import os



from google.colab import drive

drive.mount('/content/drive')
folder_path = '/content/drive/My Drive/deepl_translated_jabber'


files = os.listdir(folder_path)
print(files)
```

```
Mounted at /content/drive
['2021', '2022', '2020', 'selected_data', 'selected_modified', 'dataframetest.csv', 'p
```

```
# 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')}")
```

```
Selected files have been stored in the following directories:
2020: /content/drive/My Drive/deepl_translated_jabber/random10/2020
2021: /content/drive/My Drive/deepl_translated_jabber/random10/2021
```

```
2022: /content/drive/My Drive/deepl_translated_jabber/random10/2022
```

```
'''import os
import shutil
import random

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:
        f.write(fixed_data)

random10_dir = '/content/drive/My Drive/deepl_translated_jabber/random10'
preprocessed_data_dir = '/content/drive/My Drive/deepl_translated_jabber/preprocessedData'

os.makedirs(preprocessed_data_dir, exist_ok=True)

for year_folder in os.listdir(random10_dir):
    year_dir = os.path.join(random10_dir, year_folder)
    if os.path.isdir(year_dir):
        dest_dir = os.path.join(preprocessed_data_dir, year_folder)
        os.makedirs(dest_dir, exist_ok=True)

        for file_name in os.listdir(year_dir):
            file_path = os.path.join(year_dir, file_name)
            try:
                fix_json_format(file_path)

                dest_file_path = os.path.join(dest_dir, file_name)
                shutil.copy(file_path, dest_file_path)
                print(f"Successfully updated and copied file: {dest_file_path}")
            except Exception as e:
                print(f"Error updating file {file_path}: {e}")
'''
```

```
Successfully updated and copied file: /content/drive/My Drive/deepl_translated_jabber/
Successfully updated and copied file: /content/drive/My Drive/deepl_translated_jabber/
Successfully updated and copied file: /content/drive/My Drive/deepl_translated_jabber/
Successfully updated and copied file: /content/drive/My Drive/deepl_translated_jabber/
Successfully updated and copied file: /content/drive/My Drive/deepl_translated_jabber/
Successfully updated and copied file: /content/drive/My Drive/deepl_translated_jabber/
Successfully updated and copied file: /content/drive/My Drive/deepl_translated_jabber/
Successfully updated and copied file: /content/drive/My Drive/deepl_translated_jabber/
Successfully updated and copied file: /content/drive/My Drive/deepl_translated_jabber/
Successfully updated and copied file: /content/drive/My Drive/deepl_translated_jabber/
Successfully updated and copied file: /content/drive/My Drive/deepl_translated_jabber/
Successfully updated and copied file: /content/drive/My Drive/deepl_translated_jabber/
Successfully updated and copied file: /content/drive/My Drive/deepl_translated_jabber/
Successfully updated and copied file: /content/drive/My Drive/deepl_translated_jabber/
Successfully updated and copied file: /content/drive/My Drive/deepl_translated_jabber/
Successfully updated and copied file: /content/drive/My Drive/deepl_translated_jabber/
Successfully updated and copied file: /content/drive/My Drive/deepl_translated_jabber/
Successfully updated and copied file: /content/drive/My Drive/deepl_translated_jabber/
Successfully updated and copied file: /content/drive/My Drive/deepl_translated_jabber/
Successfully updated and copied file: /content/drive/My Drive/deepl_translated_jabber/
Successfully updated and copied file: /content/drive/My Drive/deepl_translated_jabber/
Successfully updated and copied file: /content/drive/My Drive/deepl_translated_jabber/
```

```
Successfully updated and copied file: /content/drive/My Drive/deepl_translated_jabber/
Successfully updated and copied file: /content/drive/My Drive/deepl_translated_jabber/
Successfully updated and copied file: /content/drive/My Drive/deepl_translated_jabber/
Successfully updated and copied file: /content/drive/My Drive/deepl_translated_jabber/
Successfully updated and copied file: /content/drive/My Drive/deepl_translated_jabber/
Successfully updated and copied file: /content/drive/My Drive/deepl_translated_jabber/
Successfully updated and copied file: /content/drive/My Drive/deepl_translated_jabber/
Successfully updated and copied file: /content/drive/My Drive/deepl_translated_jabber/
```

```
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())
```

```
                           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
```

```
df_combined.to_csv('/content/drive/My Drive/deepl_translated_jabber/combineddata.csv', ind
```

```
Start coding or generate with AI.
```

Data transformation

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.

```
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)
```

```
{'target@q3mcco35auwcstmt.onion': 'ID001', 'professor@q3mcco35auwcstmt.onion': 'ID002'
```

3. Concatenate the text for each individual

```
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)
```

```
23873it [00:01, 13694.06it/s]{'ID001': "hahahahahahahaha from : office asking well. ti
```

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

```
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_e
print(weighted_edges_list)
```

```
[('ID001', 'ID002', 795), ('ID002', 'ID001', 226), ('ID001', 'ID004', 456), ('ID003',
```

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

```
'''' for no 2 solution
'''
email_id_mapping = {email: f"ID{i+1:03d}" for i, email in enumerate(pd.unique(df_combined[

print(email_id_mapping)
```

```
{'target@q3mcco35auwcstmt.onion': 'ID001', 'professor@q3mcco35auwcstmt.onion': 'ID002'
```

```
'''' for no 3 solution
'''
from tqdm import tqdm

concatenated_bodies = {email_id_mapping.get(row['from']): concatenated_bodies.get(email_id

print(concatenated_bodies)
```

```
23873it [00:01, 14624.31it/s]
{'ID001': "hahahahahahahaha from : office asking well. timelines you as they say fuck
```

Visualisation and initial analysis

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

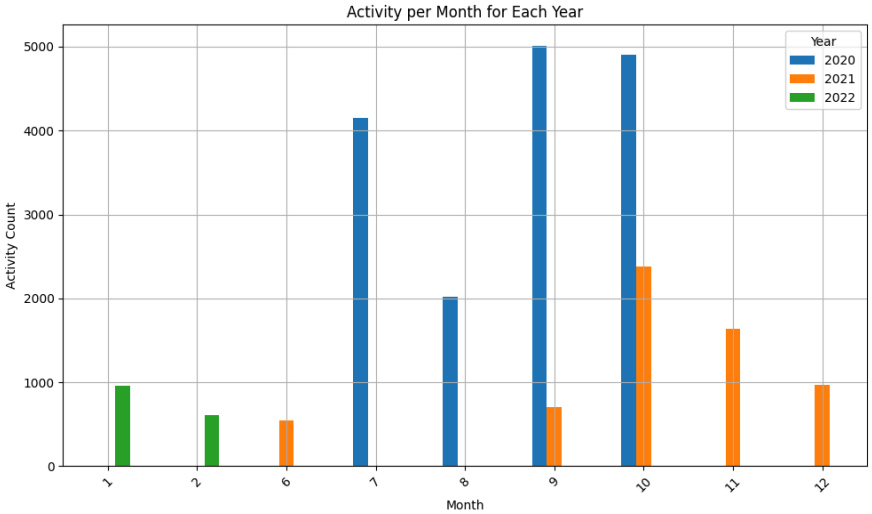
```
import pandas as pd

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', 'I

results = {}

for selected_id in selected_ids:
    selected_key = next((key for key, value in email_id_mapping.items() if value == select
    if selected_key:
        selected_member_activity = member_groups.get_group(selected_key)
        date_range = selected_member_activity['ts'].min().date(), selected_member_activity
        activity_frequency = len(selected_member_activity)
        results[selected_id] = {'date_range': date_range, 'activity_frequency': activity_f
    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")
```

```
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
```

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

```
import pandas as pd
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)
```



year	2020	2021	2022
month			
1	0	0	963
2	0	0	603
6	0	539	0
7	4148	0	0
8	2014	0	0
9	5011	707	0
10	4902	2375	0
11	0	1640	0
12	0	971	0

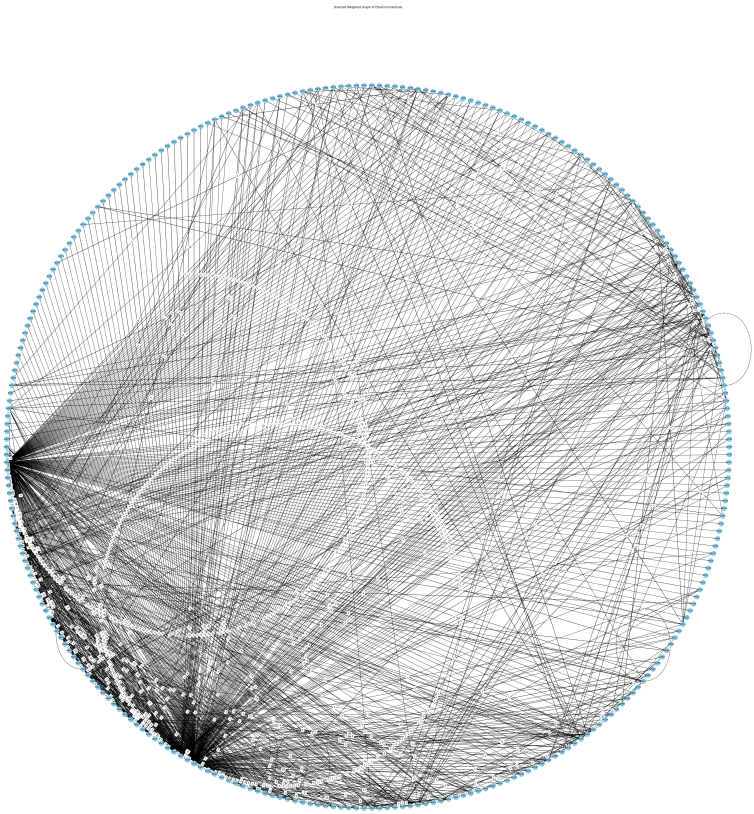
Network Analysis

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

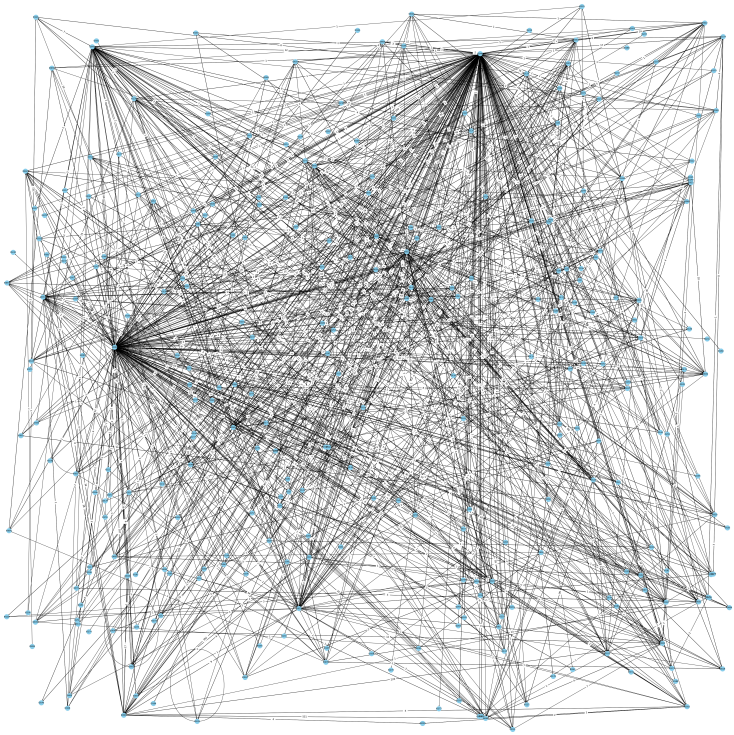
```
print(weighted_edges_list)
```

```
[('ID001', 'ID002', 795), ('ID002', 'ID001', 226), ('ID001', 'ID004', 456), ('ID003',
```

```
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)


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, arrows
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()
```

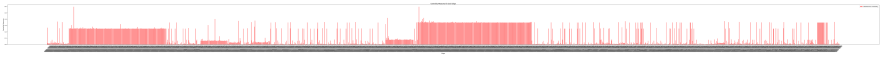


```
plt.figure(figsize=(50, 50))
pos = nx.random_layout(G)
nx.draw(G, pos, with_labels=True, node_size=400, node_color="skyblue", font_size=8, arrows
labels = nx.get_edge_attributes(G, 'weight')
nx.draw_networkx_edge_labels(G, pos, edge_labels=labels)
plt.title("Directed Weighted Graph of Email Connections")
plt.show()
```

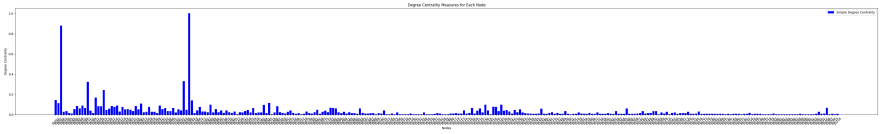


9. Calculate the centrality measures of Degree and Betweenness for each of the nodes (members) in the gang.

```
import networkx as nx
import matplotlib.pyplot as plt


#betweenness_centrality = nx.betweenness_centrality(G)
betweenness_centrality = nx.betweenness_centrality(G, normalized = True, endpoints = False
edges = list(G.edges())
betweenness_values = [betweenness_centrality[edge[0]] + betweenness_centrality[edge[1]] fo

plt.figure(figsize=(100, 6))
plt.bar(range(len(edges)), betweenness_values, color='red', label='Betweenness Centrality'
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()
```



```
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', labe
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()
```



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.

```
top_degree_members = sorted(simple_degree_centrality.items(), key=lambda x: x[1], reverse=
top_betweenness_members = sorted(betweenness_centrality.items(), key=lambda x: x[1], rever

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}")
```

```
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

Sentiment analysis

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

```
!pip install vaderSentiment
```

```
Collecting vaderSentiment
  Downloading vaderSentiment-3.3.2-py2.py3-none-any.whl (125 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 126.0/126.0 kB 3.8 MB/s eta 0:00:00
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (fr
```

```
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/d
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-pa
Installing collected packages: vaderSentiment
Successfully installed vaderSentiment-3.3.2
```

```
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)
```

```
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

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.

```
import re
lexicon = ['domain', 'server', 'encryption', 'crypt', 'malicious', 'code', 'crypto', 'file

term_frequencies = {person: {term: 0 for term in lexicon} for person in concatenated_bodie

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()
```

```
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
deploy: 0

Term frequencies for ID004:
domain: 1
server: 4
encryption: 0
crypt: 6
malicious: 0
code: 206
```

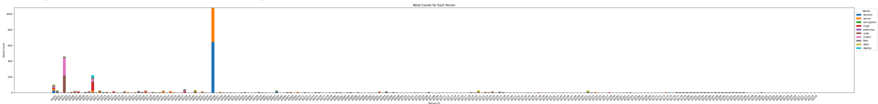
```
        crypto: 226
        files: 11
        data: 5
        deploy: 2

        Term frequencies for ID005:
        domain: 1
        server: 0
        encryption: 0
        crypt: 0
        malicious: 0
        code: 0
        crypto: 0
        files: 0
        data: 0
```

```
for i, j in concatenated_bodies.items():
  print(i + " :"+ j+ "\n")
```

```
g for you{backslash}nn_new toad domain version 3{backslash}nodw5mdwotufuxxrgw3pvqjjuz

g for you{backslash}nn_new toad domain version 3{backslash}nodw5mdwotufuxxrgw3pvqjjuz

g for you{backslash}nn_new toad domain version 3{backslash}nodw5mdwotufuxxrgw3pvqjjuz

g for you{backslash}nn_new toad domain version 3{backslash}nodw5mdwotufuxxrgw3pvqjjuz

g for you{backslash}nn_new toad domain version 3{backslash}nodw5mdwotufuxxrgw3pvqjjuz

g for you{backslash}nn_new toad domain version 3{backslash}nodw5mdwotufuxxrgw3pvqjjuz

g for you{backslash}nn_new toad domain version 3{backslash}nodw5mdwotufuxxrgw3pvqjjuz
```

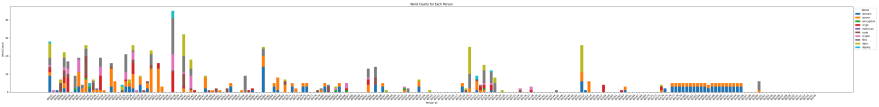
```
import matplotlib.pyplot as plt

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()
```

```
<ipython-input-24-30d7e9c264d8>:5: MatplotlibDeprecationWarning: The get_cmap functior
  legend_colors = plt.cm.get_cmap('tab10', len(lexicon))
```



```
import numpy as np

concatenated_bodies.pop('ID004', None)
concatenated_bodies.pop('ID012', None)
concatenated_bodies.pop('ID001', None)
concatenated_bodies.pop('ID046', None)
x_values = list(concatenated_bodies.keys())
legend_labels = lexicon
legend_colors = plt.cm.get_cmap('tab10', len(lexicon))
bottom = 0

plt.figure(figsize=(50, 6))
all_y_values = []

for i, term in enumerate(lexicon):
    y_values = []
    for person in concatenated_bodies:
        if person != 'ID0046':  # Skip 'ID046'
            y_values.append(term_frequencies[person][term])
    all_y_values.append(y_values)
all_y_values = np.array(all_y_values)

for i, term in enumerate(lexicon):
    plt.bar(x_values, all_y_values[i], bottom=bottom, label=term, color=legend_colors(i))
    bottom = bottom + all_y_values[i]

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()
```

```
<ipython-input-25-5ae7f26044dc>:9: MatplotlibDeprecationWarning: The get_cmap functior
  legend_colors = plt.cm.get_cmap('tab10', len(lexicon))
```



```
import nltk
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
True
```

13. **Analyse the text data.**

- **How many contain technical terms? How many technical terms are used?**

```
import nltk
from nltk.tokenize import word_tokenize
total_technical_terms = 0
technical_term_counts = {}
technical_lexicon = ["Blockchain", "Cryptocurrency", "Digital Signature", "Decentralizatio

for message in concatenated_bodies.values():
    words = word_tokenize(message)
    technical_term_count = sum(word in technical_lexicon for word in words)
    total_technical_terms += technical_term_count

print("Total technical terms used:", total_technical_terms)

for key, message in concatenated_bodies.items():
    words = word_tokenize(message)
    technical_term_count = sum(word in technical_lexicon for word in words)
    technical_term_counts[key] = technical_term_count

for key, count in technical_term_counts.items():
    print(f"Message '{key}': {count} technical terms")
```

```
Message 'ID182': 5 technical terms
Message 'ID183': 5 technical terms
Message 'ID184': 5 technical terms
Message 'ID185': 5 technical terms
Message 'ID186': 5 technical terms
Message 'ID187': 4 technical terms
Message 'ID188': 5 technical terms
Message 'ID189': 5 technical terms
Message 'ID190': 5 technical terms
Message 'ID191': 5 technical terms
Message 'ID192': 5 technical terms
Message 'ID193': 5 technical terms
Message 'ID194': 5 technical terms
Message 'ID195': 5 technical terms
Message 'ID196': 5 technical terms
Message 'ID197': 0 technical terms
Message 'ID198': 0 technical terms
Message 'ID199': 0 technical terms
Message 'ID200': 0 technical terms
Message 'ID201': 6 technical terms
Message 'ID202': 0 technical terms
Message 'ID203': 0 technical terms
Message 'ID204': 0 technical terms
Message 'ID205': 0 technical terms
Message 'ID206': 0 technical terms
Message 'ID207': 0 technical terms
```

```
# messages_with_technical_terms = sum(1 for count in technical_term_counts.values() if cou

# print("Number of messages containing technical terms:", messages_with_technical_terms)
```

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

```
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)
```

```
Complete list of terms in the data:
{'Server', 'crypto', 'Virtualization', 'Network', 'Linux', 'HTTP', 'Git', 'NFT', 'Anti
```

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

```
df_combined.head()
```

	ts	from	to	
0	2020-10-08 00:39:58.424869	target@q3mcco35auwcstmt.onion	professor@q3mcco35auwcstmt.onion	hahah
1	2020-10-08 00:40:02.225972	target@q3mcco35auwcstmt.onion	professor@q3mcco35auwcstmt.onion	

```
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)"
```

```
stern@q3mcco35auwcstmt.onion is talking technical matters with spider@q3mcco35auwcst
stern@q3mcco35auwcstmt.onion is talking technical matters with alarm@q3mcco35auwcstm
stern@q3mcco35auwcstmt.onion is talking technical matters with hors@q3mcco35auwcstmt
stern@q3mcco35auwcstmt.onion is talking technical matters with viper@q3mcco35auwcstm
stern@q3mcco35auwcstmt.onion is talking technical matters with void@q3mcco35auwcstmt
stern@q3mcco35auwcstmt.onion is talking technical matters with twin@q3mcco35auwcstmt
stern@q3mcco35auwcstmt.onion is talking technical matters with goga@q3mcco35auwcstmt
stern@q3mcco35auwcstmt.onion is talking technical matters with ahtyng@q3mcco35auwcst
stern@q3mcco35auwcstmt.onion is talking technical matters with modar@q3mcco35auwcstm
stern@q3mcco35auwcstmt.onion is talking technical matters with tatarin@q3mcco35auwcs
stern@q3mcco35auwcstmt.onion is talking technical matters with modnik@q3mcco35auwcst
stern@q3mcco35auwcstmt.onion is talking technical matters with tom@q3mcco35auwcstmt
stern@q3mcco35auwcstmt.onion is talking technical matters with beta@q3mcco35auwcstmt
stern@q3mcco35auwcstmt.onion is talking technical matters with tiktak@q3mcco35auwcst
stern@q3mcco35auwcstmt.onion is talking technical matters with dick@q3mcco35auwcstmt
stern@q3mcco35auwcstmt.onion is talking technical matters with bumer@q3mcco35auwcstm
stern@q3mcco35auwcstmt.onion is talking technical matters with rand@q3mcco35auwcstmt
stern@q3mcco35auwcstmt.onion is talking technical matters with nik-da@q3mcco35auwcst
stern@q3mcco35auwcstmt.onion is talking technical matters with sunday@q3mcco35auwcst
stern@q3mcco35auwcstmt.onion is talking technical matters with ramon@q3mcco35auwcstm
stern@q3mcco35auwcstmt.onion is talking technical matters with green@q3mcco35auwcstm
stern@q3mcco35auwcstmt.onion is talking technical matters with zulas@q3mcco35auwcstm
stern@q3mcco35auwcstmt.onion is talking technical matters with kolbasa@q3mcco35auwcs
stern@q3mcco35auwcstmt.onion is talking technical matters with kagas@q3mcco35auwcstm
stern@q3mcco35auwcstmt.onion is talking technical matters with steller@q3mcco35auwcs
stern@q3mcco35auwcstmt.onion is talking technical matters with band@q3mcco35auwcstmt
stern@q3mcco35auwcstmt.onion is talking technical matters with tunri@q3mcco35auwcstm
stern@q3mcco35auwcstmt.onion is talking technical matters with ali@q3mcco35auwcstmt
stern@q3mcco35auwcstmt.onion is talking technical matters with grant@q3mcco35auwcstm
stern@q3mcco35auwcstmt.onion is talking technical matters with hash@q3mcco35auwcstmt
stern@q3mcco35auwcstmt.onion is talking technical matters with terry@q3mcco35auwcstm
```

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

```
technical_messages = []
chitchat_messages = []

for index, row in df_combined.iterrows():
    message = row['body']

    if any(term in message.lower() for term in technical_lexicon):
        technical_messages.append(message)
    else:
        chitchat_messages.append(message)

num_technical_messages = len(technical_messages)
num_chitchat_messages = len(chitchat_messages)

print("Number of technical messages:", num_technical_messages)
print("Number of chitchat messages:", num_chitchat_messages)
```

```
Number of technical messages: 3152
Number of chitchat messages: 20721
```

```
buy_selling_lexicon = ["buy", "sell", "purchase", "sale", "price", "offer", "discount", "d
movie_discussion_lexicon = ["movie", "film", "cinema", "actor", "actress", "director", "pl
sports_lexicon = ["sports", "football", "soccer", "basketball", "tennis", "match", "game",
chitchat_lexicon = ["hi", "hello", "how are you", "what's up", "good morning", "good night
politics_lexicon = ["politics", "government", "election", "democracy", "policy", "presiden
music_lexicon = ["music", "song", "artist", "album", "band", "concert", "performance", "ge
food_lexicon = ["food", "restaurant", "cuisine", "recipe", "cook", "chef", "menu", "dish",
technology_lexicon = ["technology", "innovation", "gadget", "device", "internet", "compute
travel_lexicon = ["travel", "trip", "destination", "journey", "tourism", "adventure", "vac
books_lexicon = ["book", "novel", "author", "chapter", "fiction", "non-fiction", "literatu
current_affairs_lexicon = ["news", "current affairs", "headlines", "event", "incident", "u
personal_development_lexicon = ["personal development", "self-improvement", "growth", "mot

slangs_lexicon = [
    "af", "bae", "brb", "btw", "fomo", "ftw", "hmu", "idk", "imo", "imho", "irl", "jk", "l
    "lmfao", "lol", "omg", "rofl", "smh", "tmi", "tldr", "ttyl", "yolo", "zomg", "bff", "b
    "fud", "fudster", "fintwit", "hodl", "rekt", "stfu", "tyt", "wtf", "gtfo", "omw", "icy
    "gr8", "gn", "sry", "u", "r", "ur", "gratz", "props", "thx", "np", "yw", "ily", "xoxo"
    "gtg", "jk", "idc", "fyi", "baka", "senpai", "yandere", "tsundere", "yandere", "otaku"
    "simp", "kawaii", "savage", "stan", "ship", "op", "nvm", "dw", "tbh", "bruh", "ngl", "
    "tall", "bald", "hun", "sis", "bae", "yas", "yass", "slay", "snatched", "woke", "sksks
    "and i oop", "tea", "wig", "flex", "fam", "lit", "sus", "vibes", "mood", "cringe", "oc
    "uwu", "owo", "imao", "thicc", "pog", "pogchamp", "noob", "gg", "ez", "meme", "yeet",
    "vibing", "based", "redpilled", "based and redpilled", "based and redpilled king", "ba
"hate", "disgusting", "terrible", "awful", "crap",
"angry", "mad", "upset", "frustrated"
]

category_counts = {
    "technical": 0,
    "buy/selling": 0,
    "movie discussion": 0,
    "sports": 0,
    "chitchats": 0,
    "politics": 0,
    "music": 0,
    "food": 0,
    "technology": 0,
    "travel": 0,
    "books": 0,
    "current affairs": 0,
    "personal development": 0,
    "slangs": 0,
    "other": 0
}

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)
```

```
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
```

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

```
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, messag

    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 i

for sender, count in sorted_sender_authority_counts.items():
    print("Sender:", sender, "-->Power Pattern Count:", count)
```

```
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 -->Power 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
Sender: mushroom@q3mcco35auwcstmt.onion -->Power Pattern Count: 2
Sender: steller@q3mcco35auwcstmt.onion -->Power Pattern Count: 2
Sender: deploy@q3mcco35auwcstmt.onion -->Power Pattern Count: 2
Sender: kagas@q3mcco35auwcstmt.onion -->Power Pattern Count: 2
Sender: driver@q3mcco35auwcstmt.onion -->Power Pattern Count: 2
Sender: zulas@q3mcco35auwcstmt.onion -->Power Pattern Count: 2
Sender: van@q3mcco35auwcstmt.onion -->Power Pattern Count: 1
Sender: azot@q3mcco35auwcstmt.onion -->Power Pattern Count: 1
Sender: buza@q3mcco35auwcstmt.onion -->Power Pattern Count: 1
Sender: mavemat@q3mcco35auwcstmt.onion -->Power Pattern Count: 1
Sender: skywalker@q3mcco35auwcstmt.onion -->Power Pattern Count: 1
Sender: naned@q3mcco35auwcstmt.onion -->Power Pattern Count: 1
Sender: idgo@q3mcco35auwcstmt.onion -->Power Pattern Count: 1
Sender: mors@q3mcco35auwcstmt.onion -->Power Pattern Count: 1
Sender: baget@q3mcco35auwcstmt.onion -->Power Pattern Count: 1
Sender: electronic@q3mcco35auwcstmt.onion -->Power Pattern Count: 1
Sender: revers@q3mcco35auwcstmt.onion -->Power Pattern Count: 1
Sender: dandis@q3mcco35auwcstmt.onion -->Power Pattern Count: 1
Sender: bourbon@q3mcco35auwcstmt.onion -->Power Pattern Count: 1
Sender: kerasid@q3mcco35auwcstmt.onion -->Power Pattern Count: 1
Sender: dick@q3mcco35auwcstmt.onion -->Power Pattern Count: 1
Sender: love@q3mcco35auwcstmt.onion -->Power Pattern Count: 1
Sender: duke@q3mcco35auwcstmt.onion -->Power Pattern Count: 1
Sender: kevin@q3mcco35auwcstmt.onion -->Power Pattern Count: 1
Sender: cosmos@q3mcco35auwcstmt.onion -->Power Pattern Count: 1
Sender: zloysobaka@q3mcco35auwcstmt.onion -->Power Pattern Count: 0
Sender: troy@q3mcco35auwcstmt.onion -->Power Pattern Count: 0
Sender: many@q3mcco35auwcstmt.onion -->Power Pattern Count: 0
Sender: contisupport@q3mcco35auwcstmt.onion -->Power Pattern Count: 0
Sender: reshaev@q3mcco35auwcstmt.onion -->Power Pattern Count: 0
Sender: frog@q3mcco35auwcstmt.onion -->Power Pattern Count: 0
Sender: ganesh@q3mcco35auwcstmt.onion -->Power Pattern Count: 0
Sender: mentos@q3mcco35auwcstmt.onion -->Power Pattern Count: 0
Sender: flip@q3mcco35auwcstmt.onion -->Power Pattern Count: 0
Sender: marsel@q3mcco35auwcstmt.onion -->Power Pattern Count: 0
Sender: mavelek@q3mcco35auwcstmt.onion -->Power Pattern Count: 0
Sender: salamandra@q3mcco35auwcstmt.onion -->Power Pattern Count: 0
Sender: cany@q3mcco35auwcstmt.onion -->Power Pattern Count: 0
Sender: hash@q3mcco35auwcstmt.onion -->Power Pattern Count: 0
Sender: strix@q3mcco35auwcstmt.onion -->Power Pattern Count: 0
Sender: grant@q3mcco35auwcstmt.onion -->Power Pattern Count: 0
```

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

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

1. **neg (Negative Sentiment):** Indicates the proportion of negative words in the text. Higher values suggest a more negative tone or expression of emotions in the communication. For example: Person ID002 has a relatively high negative sentiment score (0.081), indicating a higher proportion of negative words in their communication.

2. **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.

3. **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.

4. **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?

=>

1. **Identifying Communication Patterns:** Analyzing the sentiment of messages exchanged between individuals or groups can reveal patterns in communication dynamics.
2. **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

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. **Topic Modeling:** Apply topic modeling techniques such as Latent Dirichlet Allocation (LDA) or Non-Negative Matrix Factorization (NMF) to identify the main topics discussed within the messages. This can help categorize messages into broader themes and understand the distribution of topics across different members.
3. **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.
4. **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.
5. **Deep Learning Models:** Experiment with more advanced deep learning models for sentiment analysis, to capture more nuanced sentiment expressions and improve sentiment classification accuracy.

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