Installing precursors

```
!pip install nltk
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.10/dist-packages (3.8.1)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages (from nltk) (8.1.7)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from nltk) (1.4.0)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.10/dist-packages (from nltk) (2023.12.25)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from nltk) (4.66.2)
```

Mounting Drive

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

Navigating to file and collecting data

```
import os
# Gets the contents from the file
def getJSONfiles(folder):
  path = "//content//drive//My Drive//Assignment 2//"
  diryr = path+folder+"//"
  files = os.listdir(diryr)

  files = [f for f in files if os.path.isfile(diryr + '/' + f)]
  return files


files2020 = getJSONfiles("2020")
files2021 = getJSONfiles("2021")
files2022 = getJSONfiles("2022")
```

```
import random

# Randomly selects the contents from each folder
random2020 = random.choices(files2020, k=10)
random2021 = random.choices(files2021, k=10)
random2022 = random.choices(files2022, k=10)
```

```
print(random2020[0])
```

```
185.25.51.173-20201006_en-US.json
```

Creating functions for preprocessing and json loading

```
def preprocess_json_file(file_content):

    # Split the content by '}\n{'
    json_objects = [obj.strip() for obj in file_content.split('}\n{')]

    # Add missing closing and opening braces to each JSON object
    json_objects = ['{' + obj + '}' for obj in json_objects]

    # replace double {{ and }} with single { and }, remove newlines
    for i in range(len(json_objects)):
        json_objects[i] = json_objects[i].replace('{{','{').replace('}}','}').replace('\n','')

    return json_objects


import json

def try_json_loads(obj):
  json_obj={}
  try:
    json_obj=json.loads(obj)
  except json.JSONDecodeError as e:
      print("Invalid JSON syntax", e)
      pass
  return json_obj
```

```
import pandas as pd

# Creating a process function to allow and prepare the data for analysing
def process_files_by_year(year, files):
  df_concat = pd.DataFrame()
  path = "//content//drive//My Drive//Assignment 2//deepl_translated_jabber//"
  for i in range(len(files)):
    filename = path + str(year) + "//" + files[i]

    with open(filename) as user_file:
      example_content = user_file.read()
      processed_json = preprocess_json_file(example_content)

    parsed_json_objects = [try_json_loads(obj) for obj in processed_json]
    df = pd.DataFrame(parsed_json_objects)
    df['newID'] = str(year) + "." + files[i].removesuffix('_en-US.json')
    df_concat = pd.concat([df_concat, df], axis=0)
  return df_concat


import pandas as pd
df2020 = process_files_by_year(2020, random2020)
df2021 = process_files_by_year(2021, random2021)
df2022 = process_files_by_year(2022, random2022)
df = pd.concat([df2020, df2021, df2022], axis=0)
```

```
Invalid JSON syntax Expecting ',' delimiter: line 1 column 150 (char 149)
Invalid JSON syntax Expecting ',' delimiter: line 1 column 141 (char 140)
Invalid JSON syntax Expecting ',' delimiter: line 1 column 192 (char 191)
Invalid JSON syntax Expecting ',' delimiter: line 1 column 318 (char 317)
Invalid JSON syntax Unterminated string starting at: line 1 column 132 (char 131)
Invalid JSON syntax Expecting ',' delimiter: line 1 column 164 (char 163)
Invalid JSON syntax Expecting ',' delimiter: line 1 column 170 (char 169)
Invalid JSON syntax Expecting ',' delimiter: line 1 column 182 (char 181)
Invalid JSON syntax Expecting ',' delimiter: line 1 column 167 (char 166)
Invalid JSON syntax Expecting ',' delimiter: line 1 column 157 (char 156)
Invalid JSON syntax Expecting ',' delimiter: line 1 column 142 (char 141)
Invalid JSON syntax Expecting ',' delimiter: line 1 column 142 (char 141)
Invalid JSON syntax Expecting ',' delimiter: line 1 column 555 (char 554)
Invalid JSON syntax Unterminated string starting at: line 1 column 129 (char 128)
Invalid JSON syntax Expecting ',' delimiter: line 1 column 142 (char 141)
Invalid JSON syntax Expecting ',' delimiter: line 1 column 142 (char 141)
Invalid JSON syntax Expecting ',' delimiter: line 1 column 154 (char 153)
Invalid JSON syntax Expecting ',' delimiter: line 1 column 195 (char 194)
Invalid JSON syntax Expecting ',' delimiter: line 1 column 195 (char 194)
Invalid JSON syntax Unterminated string starting at: line 1 column 127 (char 126)
Invalid JSON syntax Unterminated string starting at: line 1 column 129 (char 128)
Invalid JSON syntax Unterminated string starting at: line 1 column 132 (char 131)
Invalid JSON syntax Expecting ',' delimiter: line 1 column 146 (char 145)
Invalid JSON syntax Unterminated string starting at: line 1 column 127 (char 126)
Invalid JSON syntax Expecting ',' delimiter: line 1 column 207 (char 206)
Invalid JSON syntax Expecting ':' delimiter: line 1 column 121 (char 120)
Invalid JSON syntax Expecting ',' delimiter: line 1 column 207 (char 206)
Invalid JSON syntax Expecting ':' delimiter: line 1 column 121 (char 120)
Invalid JSON syntax Expecting ',' delimiter: line 1 column 235 (char 234)
Invalid JSON syntax Expecting ',' delimiter: line 1 column 164 (char 163)
Invalid JSON syntax Expecting ',' delimiter: line 1 column 151 (char 150)
Invalid JSON syntax Expecting ':' delimiter: line 1 column 411 (char 410)
Invalid JSON syntax Expecting ',' delimiter: line 1 column 147 (char 146)
Invalid JSON syntax Expecting ',' delimiter: line 1 column 213 (char 212)
Invalid JSON syntax Expecting ',' delimiter: line 1 column 157 (char 156)
Invalid JSON syntax Expecting ',' delimiter: line 1 column 193 (char 192)
Invalid JSON syntax Expecting ',' delimiter: line 1 column 140 (char 139)
Invalid JSON syntax Expecting ',' delimiter: line 1 column 175 (char 174)
Invalid JSON syntax Expecting ',' delimiter: line 1 column 161 (char 160)
Invalid JSON syntax Expecting ',' delimiter: line 1 column 178 (char 177)
Invalid JSON syntax Expecting ',' delimiter: line 1 column 178 (char 177)
Invalid JSON syntax Expecting ',' delimiter: line 1 column 3742 (char 3741)
Invalid JSON syntax Expecting ',' delimiter: line 1 column 300 (char 299)
Invalid JSON syntax Expecting ',' delimiter: line 1 column 1390 (char 1389)
Invalid JSON syntax Expecting ',' delimiter: line 1 column 136 (char 135)
Invalid JSON syntax Unterminated string starting at: line 1 column 127 (char 126)
Invalid JSON syntax Unterminated string starting at: line 1 column 127 (char 126)
Invalid JSON syntax Unterminated string starting at: line 1 column 127 (char 126)
Invalid JSON syntax Expecting ',' delimiter: line 1 column 694 (char 693)
Invalid JSON syntax Expecting ',' delimiter: line 1 column 696 (char 695)
Invalid JSON syntax Unterminated string starting at: line 1 column 127 (char 126)
Invalid JSON syntax Expecting ':' delimiter: line 1 column 119 (char 118)
Invalid JSON syntax Expecting ',' delimiter: line 1 column 79 (char 78)
Invalid JSON syntax Expecting ',' delimiter: line 1 column 80 (char 79)
Invalid JSON syntax Expecting ',' delimiter: line 1 column 79 (char 78)
Invalid JSON syntax Expecting ',' delimiter: line 1 column 79 (char 78)
Invalid JSON syntax Expecting ',' delimiter: line 1 column 79 (char 78)
Invalid JSON syntax Expecting ',' delimiter: line 1 column 273 (char 272)
```

```
df = pd.concat([df2020, df2021, df2022], axis=0)

# Concatenates the data so it is all in a single dataframe
```

```
print("num rows = " + str(len(df.index)))
print(df.head(5))
print(df.tail(5))
```

```
num rows = 5552
                          ts                          from  \
0  2020-10-06T00:52:56.530883  target@q3mcco35auwcstmt.onion
1  2020-10-06T00:52:57.421137  target@q3mcco35auwcstmt.onion
2  2020-10-06T00:53:00.960785  target@q3mcco35auwcstmt.onion
3  2020-10-06T00:53:02.025719  target@q3mcco35auwcstmt.onion
4  2020-10-06T00:53:03.411973  target@q3mcco35auwcstmt.onion

                                 to            body  \
0  professor@q3mcco35auwcstmt.onion            prof
1  professor@q3mcco35auwcstmt.onion           hello
2  professor@q3mcco35auwcstmt.onion  where ad_users
3  professor@q3mcco35auwcstmt.onion             all
4  professor@q3mcco35auwcstmt.onion  all in general

                        newID
0  2020.185.25.51.173-20201006
1  2020.185.25.51.173-20201006
2  2020.185.25.51.173-20201006
3  2020.185.25.51.173-20201006
4  2020.185.25.51.173-20201006
                           ts                          from  \
142  2022-01-27T20:38:26.395574  tramp@q3mcco35auwcstmt.onion
143  2022-01-27T20:38:36.181319  pumba@q3mcco35auwcstmt.onion
144  2022-01-27T20:38:37.631553  pumba@q3mcco35auwcstmt.onion
145  2022-01-27T20:40:12.436584  tramp@q3mcco35auwcstmt.onion
146  2022-01-27T20:40:20.381328  tramp@q3mcco35auwcstmt.onion

                                     to  \
142         pumba@q3mcco35auwcstmt.onion
143         tramp@q3mcco35auwcstmt.onion
144         tramp@q3mcco35auwcstmt.onion
145  cybergangster@q3mcco35auwcstmt.onion
146  cybergangster@q3mcco35auwcstmt.onion

                                          body  \
142                wait a minute ) don't drizzle )
143                                            )
144                                           ok
145                                     well yes
146  but we need to see if they came to chat or not.

                        newID
142  2022.185.25.51.173-20220127
143  2022.185.25.51.173-20220127
144  2022.185.25.51.173-20220127
145  2022.185.25.51.173-20220127
146  2022.185.25.51.173-20220127
```

```
df
```

	ts	from	
0	2020-10-06T00:52:56.530883	target@q3mcco35auwcstmt.onion	professor@q3mcco35auwcstmt.
1	2020-10-06T00:52:57.421137	target@q3mcco35auwcstmt.onion	professor@q3mcco35auwcstmt.
2	2020-10-06T00:53:00.960785	target@q3mcco35auwcstmt.onion	professor@q3mcco35auwcstmt.
3	2020-10-06T00:53:02.025719	target@q3mcco35auwcstmt.onion	professor@q3mcco35auwcstmt.
4	2020-10-06T00:53:03.411973	target@q3mcco35auwcstmt.onion	professor@q3mcco35auwcstmt.
...	...	...	
142	2022-01-27T20:38:26.395574	tramp@q3mcco35auwcstmt.onion	pumba@q3mcco35auwcstmt.

Next steps: Generate code with df View recommended plots

```
# Creating a email mapping dictionary to automatically map each individual email to each other
email_mapping = {}
id_counter = 1
for email in set(df['from']):
    email_mapping[email] = f'ID{id_counter:03}'
    id_counter += 1

for email in set(df['to']):
    if email not in email_mapping:
        email_mapping[email] = f'ID{id_counter:03}'
        id_counter += 1


df['from'] = df['from'].map(email_mapping)
df['to'] = df['to'].map(email_mapping)


df
```

	ts	from	to	body	newID
0	2020-10-06T00:52:56.530883	ID029	ID080	prof	2020.185.25.51.173-20201006
1	2020-10-06T00:52:57.421137	ID029	ID080	hello	2020.185.25.51.173-20201006
2	2020-10-06T00:53:00.960785	ID029	ID080	where ad_users	2020.185.25.51.173-20201006
3	2020-10-06T00:53:02.025719	ID029	ID080	all	2020.185.25.51.173-20201006
4	2020-10-06T00:53:03.411973	ID029	ID080	all in general	2020.185.25.51.173-20201006
...	...	...	...	...	...
142	2022-01-27T20:38:26.395574	ID032	ID006	wait a minute) don't drizzle)	2022.185.25.51.173-20220127
143	2022-01-27T20:38:36.181319	ID006	ID032	)	2022.185.25.51.173-20220127

Next steps: Generate code with df View recommended plots

```
email_mapping
```

```
        'paranoik@q3mcco35auwcstmt.onion': 'ID165',
        'ganesh@q3mcco35auwcstmt.onion': 'ID166',
        'clickclack@q3mcco35auwcstmt.onion': 'ID167',
        'tiniles@q3mcco35auwcstmt.onion': 'ID168',
        'sepvilk@q3mcco35auwcstmt.onion': 'ID169',
        'sentinel@q3mcco35auwcstmt.onion': 'ID170',
        'xenon@q3mcco35auwcstmt.onion': 'ID171',
        'blackjob@q3mcco35auwcstmt.onion': 'ID172',
        'steve@q3mcco35auwcstmt.onion': 'ID173',
        'voron@q3mcco35auwcstmt.onion': 'ID174',
        'tunri@q3mcco35auwcstmt.onion': 'ID175',
        'zevs@q3mcco35auwcstmt.onion': 'ID176',
        'max@q3mcco35auwcstmt.onion': 'ID177',
        'miguel@q3mcco35auwcstmt.onion': 'ID178',
        'barmen@q3mcco35auwcstmt.onion': 'ID179',
        'tort@q3mcco35auwcstmt.onion': 'ID180',
        'kramer@q3mcco35auwcstmt.onion': 'ID181',
        'netman@q3mcco35auwcstmt.onion': 'ID182'}
```

```
import networkx as nx

# Using directed graph to show the flow of information between users
G = nx.DiGraph()

# Adds nodes for each email id to the graph
for email, node_id in email_mapping.items():
    G.add_node(node_id, email=email)

# Iterating over rows in the dataframe, while mapping emails to IDs, than checking and updating edge weights for the graph
for _, row in df.iterrows():
    from_id = row['from']
    to_id = row['to']
    if G.has_edge(from_id, to_id):
        G[from_id][to_id]['weight'] += 1
    else:
        G.add_edge(from_id, to_id, weight=1)
```

```
print(G)
```

```
 DiGraph with 182 nodes and 618 edges
```

```
for from_id, to_id, data in G.edges(data=True):
    weight = data['weight']
    print(f"{from_id}:{to_id}, {weight}")
```

```
ID154:ID123, 15
ID154:ID045, 1
ID154:ID092, 3
ID154:ID102, 1
ID154:ID038, 2
ID154:ID150, 1
ID154:ID061, 4
ID154:ID110, 1
ID154:ID131, 3
ID154:ID060, 1
ID155:ID132, 2
ID156:ID145, 1
ID156:ID057, 1
ID156:ID090, 1
ID157:ID156, 1
ID157:ID089, 12
ID157:ID049, 2
```

```
weights = [(from_id, to_id, data['weight']) for from_id, to_id, data in G.edges(data=True)]
weights
```

```
('ID145', 'ID132', 36),
('ID145', 'ID166', 2),
('ID145', 'ID104', 3),
('ID145', 'ID135', 1),
('ID145', 'ID013', 3),
('ID145', 'ID055', 5),
('ID145', 'ID118', 16),
('ID145', 'ID128', 4),
('ID145', 'ID038', 124),
('ID145', 'ID099', 43),
('ID145', 'ID101', 4),
('ID145', 'ID060', 6),
('ID145', 'ID122', 22),
('ID145', 'ID018', 1),
('ID145', 'ID180', 7),
('ID146', 'ID025', 1),
('ID147', 'ID052', 1),
('ID148', 'ID145', 8),
('ID148', 'ID165', 2),
('ID149', 'ID089', 14),
('ID149', 'ID132', 11),
('ID149', 'ID092', 13),
('ID149', 'ID027', 1),
('ID149', 'ID131', 2),
('ID149', 'ID007', 1),
('ID149', 'ID099', 1),
('ID149', 'ID138', 2),
('ID149', 'ID148', 2),
('ID149', 'ID020', 1),
('ID149', 'ID160', 2),
('ID149', 'ID081', 2),
('ID149', 'ID077', 2),
('ID150', 'ID123', 1),
('ID150', 'ID101', 2),
('ID150', 'ID039', 1),
('ID151', 'ID145', 21),
('ID151', 'ID007', 5),
('ID152', 'ID033', 54),
('ID152', 'ID134', 1),
('ID153', 'ID104', 1),
('ID154', 'ID001', 20),
('ID154', 'ID123', 15),
('ID154', 'ID045', 1),
('ID154', 'ID092', 3),
('ID154', 'ID102', 1),
('ID154', 'ID038', 2),
('ID154', 'ID150', 1),
('ID154', 'ID061', 4),
('ID154', 'ID110', 1),
('ID154', 'ID131', 3),
('ID154', 'ID060', 1),
('ID155', 'ID132', 2),
('ID156', 'ID145', 1),
('ID156', 'ID057', 1),
('ID156', 'ID090', 1),
('ID157', 'ID156', 1),
('ID157', 'ID089', 12),
('ID157', 'ID049', 2)]
```

```
df['ts']
```

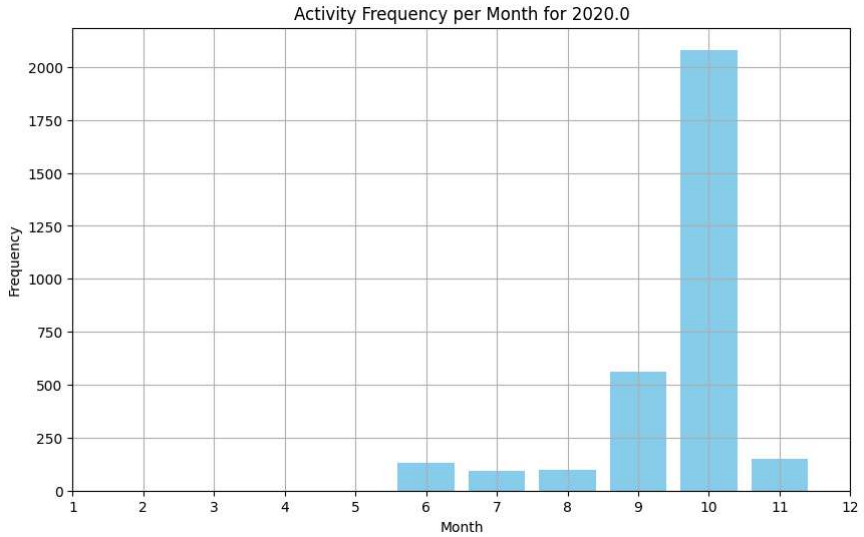
```
0      2020-10-06T00:52:56.530883
1      2020-10-06T00:52:57.421137
2      2020-10-06T00:53:00.960785
3      2020-10-06T00:53:02.025719
4      2020-10-06T00:53:03.411973
                  ...
142    2022-01-27T20:38:26.395574
```

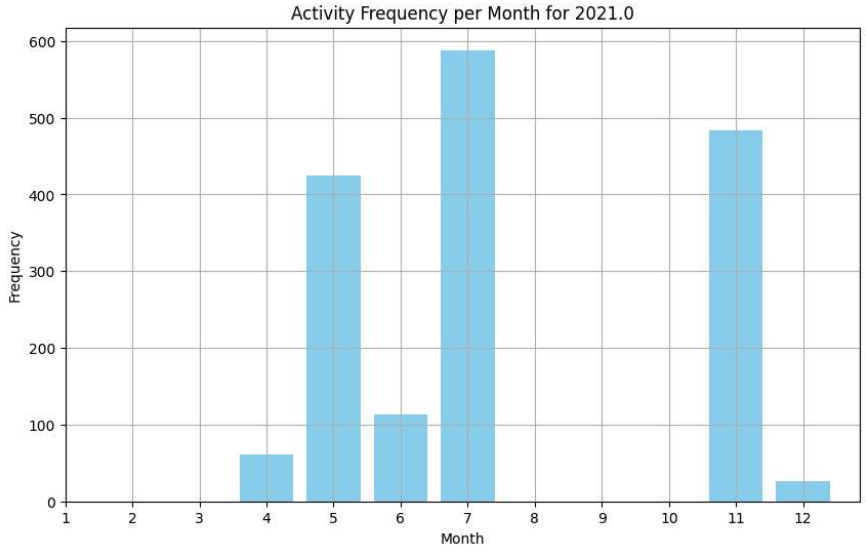
```
143    2022-01-27T20:38:36.181319
144    2022-01-27T20:38:37.631553
145    2022-01-27T20:40:12.436584
146    2022-01-27T20:40:20.381328
Name: ts, Length: 5552, dtype: object
```

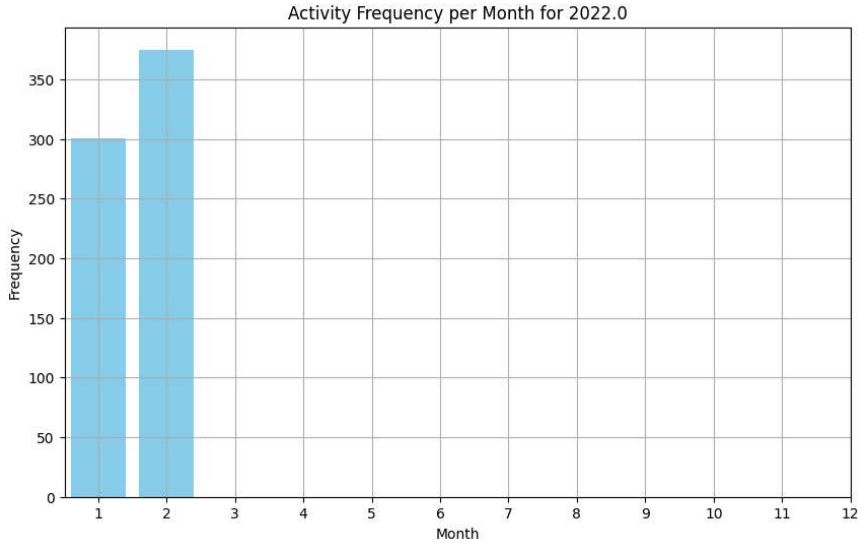
```
grouped_data = df.groupby('from')['ts']
date_ranges = grouped_data.agg(['min', 'max'])
activity_frequency = grouped_data.count()

print(date_ranges.head(5))
print(date_ranges.tail(5))
print(activity_frequency)
```

```
                                min                         max
from
ID001  2020-08-23T14:06:08.229765  2020-11-11T22:21:24.991083
ID002  2020-09-05T14:01:12.608189  2020-09-05T14:01:12.608189
ID003  2020-07-04T08:31:29.529840  2022-01-27T19:12:41.296782
ID004  2020-10-13T08:48:07.069411  2020-10-13T08:48:08.975005
ID005  2021-05-21T08:49:02.924472  2021-05-21T09:55:08.657831
                                min                         max
from
ID153  2021-11-19T16:14:26.484430  2021-11-19T16:14:26.484430
ID154  2020-10-06T13:47:52.824879  2022-01-18T21:19:42.169420
ID155  2020-11-11T11:33:27.671196  2020-11-11T11:44:30.666178
ID156  2020-10-06T11:22:27.880616  2021-05-12T10:03:15.683479
ID157  2020-10-06T04:45:56.262745  2022-02-23T13:52:12.251331
from
ID001    102
ID002      1
ID003     20
ID004      2
ID005      5
         ...
ID153      1
ID154     52
ID155      2
ID156      3
ID157     15
Name: ts, Length: 157, dtype: int64
```

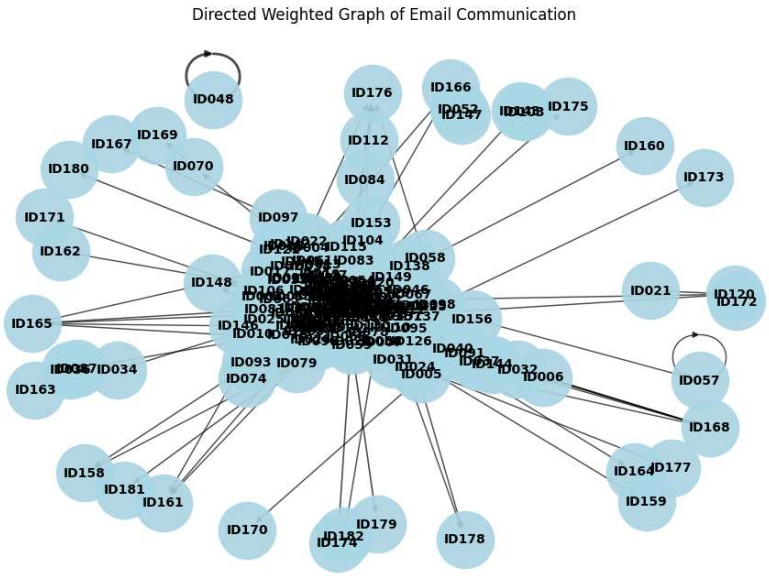
```
import matplotlib.pyplot as plt

# Converting the ts into datetime
df['ts'] = pd.to_datetime(df['ts'])

# Creating a variable for the year and month
df['year'] = df['ts'].dt.year
df['month'] = df['ts'].dt.month

# Grouping the data by year and month, while also counting their activity during that time
activity_per_month = df.groupby(['year', 'month']).size().reset_index(name='activity_count')

# Plotting a histogram for each year to project the activity for each month
years = activity_per_month['year'].unique()
for year in years:
    data_year = activity_per_month[activity_per_month['year'] == year]
    plt.figure(figsize=(10, 6))
    plt.bar(data_year['month'], data_year['activity_count'], color='skyblue')
    plt.title(f'Activity Frequency per Month for {year}')
    plt.xlabel('Month')
    plt.ylabel('Frequency')
    plt.xticks(range(1, 13))
    plt.grid(True)
    plt.show()
```


Activity Frequency per Month for 2020.0
Frequency
Month
0
250
500
750
1000
1250
1500
1750
2000
1
2
3
4
5
6
7
8
9
10
11
12


Activity Frequency per Month for 2021.0
Frequency
Month
0
100
200
300
400
500
600
1
2
3
4
5
6
7
8
9
10
11
12


Activity Frequency per Month for 2022.0
Frequency
Month
0
50
100
150
200
250
300
350
1
2
3
4
5
6
7
8
9
10
11
12

```
G = nx.DiGraph()

# Add nodes for each email ID to the graph
for from_id, to_id, _ in weights:
    G.add_node(from_id)
    G.add_node(to_id)

# Add weighted edges between nodes representing email communication
for from_id, to_id, weight in weights:
    G.add_edge(from_id, to_id, weight=weight)

# Draw the graph
plt.figure(figsize=(12, 8))
pos = nx.spring_layout(G)  # Position nodes using a spring layout algorithm

# Calculate edge thickness based on the weight
edge_weights = [data['weight'] for _, _, data in G.edges(data=True)]
max_weight = max(edge_weights)
min_weight = min(edge_weights)
edge_thickness = [(weight - min_weight) / (max_weight - min_weight) * 5 + 1 for weight in edge_weights]

# Draw edges with thickness proportional to the weight
nx.draw_networkx_edges(G, pos, width=edge_thickness, alpha=0.7)

# Draw nodes with labels
nx.draw_networkx_nodes(G, pos, node_color='lightblue', node_size=2000, alpha=0.9)
nx.draw_networkx_labels(G, pos, font_size=10, font_weight='bold')

plt.title('Directed Weighted Graph of Email Communication')
plt.axis('off')
plt.show()
```




```
degree_centrality = nx.degree_centrality(G)
degree_centrality
```

```
 'ID063': 0.016574585635359115,
 'ID064': 0.016574585635359115,
 'ID065': 0.016574585635359115,
 'ID066': 0.022099447513812154,
 'ID073': 0.049723756906077346,
 'ID068': 0.011049723756906077,
 'ID069': 0.04419889502762431,
 'ID129': 0.016574585635359115,
 'ID169': 0.0055248618784530384,
 'ID071': 0.011049723756906077,
 'ID156': 0.03314917127071823,
 'ID074': 0.011049723756906077,
 'ID075': 0.0055248618784530384,
 'ID100': 0.016574585635359115,
 'ID077': 0.022099447513812154,
 'ID078': 0.011049723756906077,
 'ID090': 0.022099447513812154,
 'ID158': 0.011049723756906077,
 'ID181': 0.0055248618784530384,
 'ID113': 0.022099447513812154,
 'ID135': 0.022099447513812154,
 'ID114': 0.027624309392265192,
 'ID084': 0.011049723756906077,
 'ID112': 0.016574585635359115,
 'ID088': 0.011049723756906077,
 'ID182': 0.0055248618784530384,
 'ID137': 0.016574585635359115,
 'ID111': 0.022099447513812154,
 'ID164': 0.0055248618784530384,
 'ID097': 0.016574585635359115,
 'ID167': 0.0055248618784530384,
 'ID098': 0.011049723756906077,
 'ID103': 0.011049723756906077,
 'ID143': 0.016574585635359115,
 'ID166': 0.011049723756906077,
 'ID105': 0.016574585635359115,
 'ID109': 0.0055248618784530384,
 'ID173': 0.0055248618784530384,
 'ID117': 0.016574585635359115,
 'ID172': 0.0055248618784530384,
 'ID124': 0.016574585635359115,
 'ID125': 0.0055248618784530384,
 'ID126': 0.0055248618784530384,
 'ID155': 0.011049723756906077,
 'ID162': 0.0055248618784530384,
 'ID175': 0.0055248618784530384,
 'ID133': 0.016574585635359115,
 'ID134': 0.022099447513812154,
 'ID136': 0.0055248618784530384,
 'ID142': 0.0055248618784530384,
 'ID180': 0.0055248618784530384,
 'ID160': 0.0055248618784530384,
 'ID153': 0.0055248618784530384}
```

```
betweenness_centrality = nx.betweenness_centrality(G)
betweenness_centrality
```

```
 'ID164': 0.0,
 'ID097': 0.0,
 'ID167': 0.0,
 'ID098': 0.0,
 'ID103': 0.0,
 'ID143': 0.004389195825659914,
 'ID166': 0.0,
 'ID105': 0.00020887457441625112,
 'ID109': 0.0,
 'ID173': 0.0,
 'ID117': 0.0,
 'ID172': 0.0,
 'ID124': 0.0008650739506366145,
 'ID125': 0.0,
 'ID126': 0.0,
 'ID155': 0.0,
 'ID162': 0.0,
 'ID175': 0.0,
 'ID133': 0.0,
 'ID134': 0.0021236614146374734,
 'ID136': 0.0,
 'ID142': 0.0,
 'ID180': 0.0,
 'ID160': 0.0,
 'ID153': 0.0}
```

```
high_degree_nodes = [node for node, centrality in degree_centrality.items() if centrality > 0.1]

high_betweenness_nodes = [node for node, centrality in betweenness_centrality.items() if centrality > 0.1]
```

```
print("Nodes with high degree centrality:", high_degree_nodes)
print("Nodes with high betweenness centrality", high_betweenness_nodes)

# Looks through all of the emails and finds the ones that have the highest betweenness and degree, to assist in determining the most imp
important_members = set(high_degree_nodes).intersection(high_betweenness_nodes)
```

```
Nodes with high degree centrality: ['ID001', 'ID089', 'ID154', 'ID132', 'ID104', 'ID080', 'ID040', 'ID007', 'ID039', 'ID029', 'ID083
Nodes with high betweenness centrality ['ID089', 'ID132', 'ID039']
```

```
print('The most importand members at are: ')
for node in important_members:
  print(node)
```

```
The most importand members at 10% are:
ID132
ID089
ID039
```

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

```
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
True
```

```
from nltk.sentiment.vader import SentimentIntensityAnalyzer

# Setting up the analyzer for use
analyzer = SentimentIntensityAnalyzer()
```

```
analyzer.polarity_scores(str(grouped_data))
```

```
{'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0}
```

```
grouped_data = df.groupby('body').agg(lambda x: ''.join(str(e) for e in set(x))).reset_index()
```

```
grouped_data
```

	body	ts	from
0	2020-10-13{backslash}n[01:16:31] <green> this...	2020-10-13 03:08:56.936639	ID089
1	<bentley> ready {backslash}n[14:34:00] <bentl...	2022-01-02 11:25:18.361651	ID089
2	<bill> 08(&JRH^EG%X$5y3tc4y9v50tb8YM()KT(78rn...	2020-10-06 08:35:50.097957	ID089
3	<doomsday> testlab{backslash}{backslash}jschi...	2020-10-06 14:37:20.024220	ID086
4	Download: https://qaz.im/load/43GGB9/NtHHGh {...	2020-09-05 12:53:49.8037632020-09-05 10:17:01....	ID029
...	...	...	..
3777	{backslash}you better tell me how to typedef f...	2020-11-11 09:01:10.818599	ID152
		2020-10-13	

Next steps: Generate code with grouped_data View recommended plots

```
def calculate_sentiment(text):
    return analyzer.polarity_scores(text)


# prompt: Using grouped_data, do a sentiment analysis of each email

grouped_data['sentiment'] = grouped_data['body'].apply(calculate_sentiment)
grouped_data.head()
```

	body	ts	from	to	
0	2020-10-13{backslash}n[01:16:31] <green> this...	2020-10-13 03:08:56.936639	ID089	ID157	2020.18
1	<bentley> ready {backslash}n[14:34:00] <bentl...	2022-01-02 11:25:18.361651	ID089	ID164	2022.18

Next steps: Generate code with grouped_data View recommended plots

```
# Define a list of technical terms related to ransomware scripts
technical_terms = [
    ".conti",
    "network",
    "bitcoin",
    "payment",
    "malware",
    "encrypted",
    "decrypt",
    "payload",
    "antivirus",
    "proof"
]


# Create a function to count the occurrences of technical terms in a text
def count_technical_terms(text):
    count = 0
    for term in technical_terms:
        if term in text.lower():
            count += 1
    return count

# Apply the function to the 'body' column of the grouped_data DataFrame
grouped_data['technical_term_count'] = grouped_data['body'].apply(count_technical_terms)

# Print the head and tail of the grouped_data DataFrame
print(grouped_data.head())
print(grouped_data.tail())
```

```
4   Download: https://qaz.im/load/43GGB9/NtHHGn {...

                                                ts   from     to  \
0                        2020-10-13 03:08:56.936639  ID089  ID157
1                        2022-01-02 11:25:18.361651  ID089  ID164
2                        2020-10-06 08:35:50.097957  ID089  ID073
3                        2020-10-06 14:37:20.024220  ID086  ID113
4  2020-09-05 12:53:49.8037632020-09-05 10:17:01....  ID029  ID089

                          newID    year month  \
0  2020.185.25.51.173-20201013  2020.0  10.0
1  2022.185.25.51.173-20220102  2022.0   1.0
2  2020.185.25.51.173-20201006  2020.0  10.0
3  2020.185.25.51.173-20201006  2020.0  10.0
4  2020.185.25.51.173-20200905  2020.0   9.0

                                          sentiment  technical_term_count
0  {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound...                     0
1  {'neg': 0.0, 'neu': 0.737, 'pos': 0.263, 'comp...                     0
2  {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound...                     0
3  {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound...                     0
4  {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound...                     0
                                                body  \
3777  {backslash}you better tell me how to typedef f...
3778                 {backslash}{backslash}{backslash}
3779  Привет{backslash}n{backslash}nbot{backslash}n{...
3780                                            отово
3781                                                😊

                                                  ts             from  \
3777                         2020-11-11 09:01:10.818599           ID152
3778  2020-10-13 15:33:09.0440022020-07-11 17:52:37....  ID065ID082ID012
3779  2020-10-13 06:06:15.8861602020-10-13 07:29:05....           ID104
3780                         2020-10-06 13:22:47.451144           ID062
3781  2020-09-26 19:29:23.7537642020-09-26 19:31:52....           ID059

                    to                                          newID  \
3777            ID033                       2020.185.25.51.173-20201111
3778  ID035ID107ID132  2021.185.25.51.173-202107162020.185.25.51.173-...
3779       ID007ID127                       2020.185.25.51.173-20201013
3780            ID096                       2020.185.25.51.173-20201006
3781            ID029                       2020.185.25.51.173-20200926

                year    month  \
3777        2020.0     11.0
3778  2020.02021.0  10.07.0
3779        2020.0     10.0
3780        2020.0     10.0
3781        2020.0      9.0

                                           sentiment  technical_term_count
3777  {'neg': 0.0, 'neu': 0.838, 'pos': 0.162, 'comp...                     0
3778  {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound...                     0
3779  {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound...                     0
3780  {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound...                     0
3781  {'neg': 0.0, 'neu': 0.0, 'pos': 0.0, 'compound...                     0
```

```
# Prints all emails that contain technical terms and their technical term count

technical_emails = grouped_data[grouped_data['technical_term_count'] > 0]
for _, row in technical_emails.iterrows():
    print(f"Email: {row['to']}")
    print(f"Technical term count: {row['technical_term_count']}")
    print(f"Body: {row['body']}")
    print(f"Terms: {', '.join([term for term in technical_terms if term in row['body'].lower()])}")
    print("--------------------")
```

```
------------------
Email: ID035
Technical term count: 1
Body: skimmed you the offdecryptor, if you don't accept it the helpdesk will be waiting for you just in case
Terms: decrypt
------------------
Email: ID132
Technical term count: 1
Body: systemtechnologyinc.com{backslash}n3 SERWA, 20 WORKs{backslash}nMega{backslash}norangeapple341@mail.com{backslash}norange32
Terms: payment
------------------
Email: ID144
Technical term count: 1
Body: that's it, the end of the bitcoins
Terms: bitcoin
------------------
Email: ID132
Technical term count: 1
Body: two questions {backslash}n1) I'm using your antivirus there - and I ran out of licenses - please extend it, I don't use it
Terms: antivirus
------------------
Email: ID029
Technical term count: 1
Body: well, he had payments.
Terms: payment
------------------
Email: ID152
Technical term count: 1
Body: you can overdo it and it will look suspicious to antivirus.
Terms: antivirus
------------------
Email: ID131
Technical term count: 1
Body: your messages are encrypted again, restart psi, write again
Terms: encrypted
------------------
```

```
# Create a dictionary to store the counts of each technical term
term_counts = {}

# Loop through the technical terms
for term in technical_terms:
    term_counts[term] = 0

# Loop through the rows of the grouped_data DataFrame
for _, row in grouped_data.iterrows():
    body = row['body']

    # Check if the term is present in the body
    for term in technical_terms:
        if term in body.lower():
            term_counts[term] += 1

total_technical_terms = sum(term_counts.values())
technical_emails_count = grouped_data[grouped_data['technical_term_count'] > 0].shape[0]
```

```
print(f"In total there are {total_technical_terms} tecnical terms found within {technical_emails_count} emails")
# Print the count of each term
print("These are the terms found:")
for term, count in term_counts.items():
    print(f"{term}: {count}")
```

```
In total there are 50 tecnical terms found within 43 emails
These are the terms found:
.conti: 0
network: 4
bitcoin: 4
payment: 11
malware: 0
encrypted: 6
decrypt: 20
payload: 0
antivirus: 3
proof: 2
```

Using this code allows to find the senders and recipients of these key terms

```
# Creates a dictionary to store the counts of each technical term
term_counts = {}

# Loops through the technical terms
for term in technical terms:
```

```
    term_counts[term] = 0

# Loops through the rows of the grouped_data DataFrame
for _, row in grouped_data.iterrows():
    body = row['body']

    # Checking if a term is present in the body
    for term in technical_terms:
        if term in body.lower():
            term_counts[term] += 1

# Creates a dictionary to store the senders and recipients of technical emails
sender_recipient_dict = {}

# Loops through the rows of the grouped_data DataFrame
for _, row in grouped_data.iterrows():
    body = row['body']

    # This is checking if the emails have any of the technical terms
    if any(term in body.lower() for term in technical_terms):
        # Once they find it, they retrieve who the senders and recipients of these technical terms are
        sender = row['from']
        recipient = row['to']

        # Adds the sender and recipients to the dictionary
        if sender not in sender_recipient_dict:
            sender_recipient_dict[sender] = []
        sender_recipient_dict[sender].append(recipient)

# Prints the senders and recipients of emails
for sender, recipients in sender_recipient_dict.items():
    print(f"Sender: {sender}")
    print(f"Recipients: {recipients}")
    print("--------------------")
```

```
Sender: ID024
Recipients: ['ID056', 'ID056']
--------------------
Sender: ID087
Recipients: ['ID036', 'ID036', 'ID036']
--------------------
Sender: ID037
Recipients: ['ID168', 'ID168', 'ID144', 'ID144', 'ID040', 'ID040', 'ID040', 'ID144']
--------------------
Sender: ID085
Recipients: ['ID003']
--------------------
Sender: ID030
Recipients: ['ID039']
--------------------
Sender: ID145
Recipients: ['ID132', 'ID132']
--------------------
Sender: ID132
Recipients: ['ID080', 'ID145', 'ID029']
--------------------
Sender: ID006
Recipients: ['ID144', 'ID032', 'ID040ID168']
--------------------
Sender: ID039
Recipients: ['ID132', 'ID132']
--------------------
Sender: ID016
Recipients: ['ID132', 'ID132']
--------------------
Sender: ID128
Recipients: ['ID145']
--------------------
Sender: ID118ID038ID145
Recipients: ['ID038ID145ID118ID099']
--------------------
Sender: ID038ID145ID099ID118
Recipients: ['ID038ID128ID099ID145ID118']
--------------------
Sender: ID080
Recipients: ['ID132', 'ID081', 'ID132', 'ID165', 'ID132']
--------------------
Sender: ID144
Recipients: ['ID037']
--------------------
Sender: ID144ID006
Recipients: ['ID144ID006']
--------------------
Sender: ID001
```

```
Recipients: ['ID132']
------------------
Sender: ID035
Recipients: ['ID081']
------------------
Sender: ID081
Recipients: ['ID035', 'ID035']
------------------
```

There are various more experiments that could be done to better understand the data, some include using machine learning to analyse the data, others could be conducting a network analysis to determine the various patterns and relationships between different components of the infestructure.

The components of data analytics process is firstly, obtaining the data. Next would be to do data cleaning and preprocessing the dataset and