Response: Summary and Observation

I used a 1% random sample of values for most of the analysis. Out of the six metrics, K-Core Centrality and PageRank are the most interesting. The K-Core Centrality histogram shows that most nodes are in the outer layers of the network, with only a small number deeply embedded in the core. This means the network relies on a few highly connected nodes to stay strong. PageRank helps pinpoint the most influential nodes, especially in the high-replies dataset, where a small number of nodes dominate.

Degree Centrality and Betweenness Centrality are helpful but harder to interpret visually because the graphs have overlapping labels. Degree Centrality shows the nodes with the most connections, while Betweenness Centrality highlights the ones that act as bridges connecting different parts of the network. Closeness Centrality shows how easily nodes can reach others in the network, and Percolation Centrality reveals that most nodes don't have much impact on the overall structure.

In summary, the analysis shows that the network depends on a few key nodes for strength and influence, while most nodes play a less significant role.

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

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

```
Mounted at /content/drive
```

```
import pandas as pd
import networkx as nx
import numpy as np
%matplotlib inline
```

```
!ls "/content/drive/My Drive/Colab Notebooks/cs131"
```

```
cleaned_combined_AdjList.tsv                         replies_nobots_uniq_lowinfluence.txt
combined_AdjList.tsv                                 retweets_highinfluence_count.txt
Copy_of_twitter_high_and_low_analysis_2024-1.ipynb  retweets_lowinfluence_count.txt
highinfl_AdjList.tsv                                 retweets_nobots_uniq_highinfluence.NONOUSER.txt
lowinfl_AdjList.tsv                                  retweets_nobots_uniq_highinfluence.txt
replies_highinfluence_count.txt                      retweets_nobots_uniq_lowinfluence.NONOUSER.txt
replies_lowinfluence_count.txt                       retweets_nobots_uniq_lowinfluence.txt
replies_nobots_uniq_highinfluence.txt
```

```
#DEFINE COLUMN
column_names=['A','B']
```

```
# Load the data
highinfl_df = pd.read_csv('/content/drive/My Drive/Colab Notebooks/cs131/retweets_nobots_uniq_highinfluence.txt',
                          sep=r'\s+', names=column_names, engine='python')
lowinfl_df = pd.read_csv('/content/drive/My Drive/Colab Notebooks/cs131/retweets_nobots_uniq_lowinfluence.txt',
                         sep=r'\s+', names=column_names, engine='python')

# Ensure columns 'A' and 'B' are strings
highinfl_df['A'] = highinfl_df['A'].astype(str)
highinfl_df['B'] = highinfl_df['B'].astype(str)

lowinfl_df['A'] = lowinfl_df['A'].astype(str)
lowinfl_df['B'] = lowinfl_df['B'].astype(str)

# Clean each DataFrame by removing rows with 'UNKNOWNUSER' in columns A or B
highinfl_df = highinfl_df[~highinfl_df['A'].str.contains(r'^UNKNOWNUSER\d+', na=False)]
highinfl_df = highinfl_df[~highinfl_df['B'].str.contains(r'^UNKNOWNUSER\d+', na=False)]

lowinfl_df = lowinfl_df[~lowinfl_df['A'].str.contains(r'^UNKNOWNUSER\d+', na=False)]
lowinfl_df = lowinfl_df[~lowinfl_df['B'].str.contains(r'^UNKNOWNUSER\d+', na=False)]

# Save the cleaned individual DataFrames
highinfl_df.to_csv('/content/drive/My Drive/Colab Notebooks/cs131/retweets_nobots_uniq_highinfluence.NONOUSER.txt',
                   sep='\t', index=False, header=False)
lowinfl_df.to_csv('/content/drive/My Drive/Colab Notebooks/cs131/retweets_nobots_uniq_lowinfluence.NONOUSER.txt',
                  sep='\t', index=False, header=False)

# Combine the cleaned DataFrames
combined_data = pd.concat([highinfl_df, lowinfl_df], ignore_index=True)

# Save the combined data
combined_data.to_csv('/content/drive/My Drive/Colab Notebooks/cs131/cleaned_combined_AdjList.tsv',
                     sep='\t', index=False)

print("Cleaning and saving completed successfully!")
```

```
Cleaning and saving completed successfully!
```

```
# Define column names
column_names = ['A', 'B']

# Load the data
highinfl_df = pd.read_csv('/content/drive/My Drive/Colab Notebooks/cs131/replies_nobots_uniq_highinfluence.txt',
                          sep=r'\s+', names=column_names, engine='python')
lowinfl_df = pd.read_csv('/content/drive/My Drive/Colab Notebooks/cs131/replies_nobots_uniq_lowinfluence.txt',
                         sep=r'\s+', names=column_names, engine='python')

# Ensure columns 'A' and 'B' are strings
highinfl_df['A'] = highinfl_df['A'].astype(str)
highinfl_df['B'] = highinfl_df['B'].astype(str)

lowinfl_df['A'] = lowinfl_df['A'].astype(str)
lowinfl_df['B'] = lowinfl_df['B'].astype(str)

# Clean each DataFrame by removing rows with 'UNKNOWNUSER' in columns A or B
highinfl_df = highinfl_df[~highinfl_df['A'].str.contains(r'^UNKNOWNUSER\d+', na=False)]
highinfl_df = highinfl_df[~highinfl_df['B'].str.contains(r'^UNKNOWNUSER\d+', na=False)]

lowinfl_df = lowinfl_df[~lowinfl_df['A'].str.contains(r'^UNKNOWNUSER\d+', na=False)]
lowinfl_df = lowinfl_df[~lowinfl_df['B'].str.contains(r'^UNKNOWNUSER\d+', na=False)]

# Save the cleaned individual DataFrames
highinfl_df.to_csv('/content/drive/My Drive/Colab Notebooks/cs131/replies_nobots_uniq_highinfluence.NONOUSER.txt',
                   sep='\t', index=False, header=False)
lowinfl_df.to_csv('/content/drive/My Drive/Colab Notebooks/cs131/replies_nobots_uniq_lowinfluence.NONOUSER.txt',
                  sep='\t', index=False, header=False)
```

```
# Combine the cleaned DataFrames
combined_data = pd.concat([highinfl_df, lowinfl_df], ignore_index=True)

# Save the combined data
combined_data.to_csv('/content/drive/My Drive/Colab Notebooks/cs131/cleaned_replies_combined_AdjList.tsv',
                     sep='\t', index=False)

print("Cleaning and saving completed successfully!")
```

```
Cleaning and saving completed successfully!
```

```
highinfl_df.head()
```

	A	B
0	100256670	105327432
1	100256670	1058038885765255175
2	100256670	1064545651
3	100256670	109071031
4	100256670	1120698997327294464

Next steps: Generate code with highinfl_df View recommended plots New interactive sheet

```
#print the number of columns in the data frame
num_rows,num_cols=combined_data.shape
print('Number of rows:',num_rows)
print('Number of column:',num_cols)
```

```
Number of rows: 88718
Number of column: 2
```

```
#How many number of rows and columns for retweeet
num_rows,num_cols=highinfl_df.shape
print('Number of rows:',num_rows)
print('Number of column:',num_cols)
```

```
Number of rows: 42426
Number of column: 2
```

```
#How many number of rows and columns for retweeet
num_rows,num_cols=lowinfl_df.shape
print('Number of rows:',num_rows)
print('Number of column:',num_cols)
```

```
Number of rows: 46292
Number of column: 2
```

```
highinfl_df['B']=highinfl_df['B'].astype(int)
lowinfl_df['B']=lowinfl_df['B'].astype(int)
```

```
#view the first few rows and columns for high for retweet
highinfl_df.head()
```

	A	B
0	100256670	105327432
1	100256670	1058038885765255175
2	100256670	1064545651
3	100256670	109071031
4	100256670	1120698997327294464

Next steps: Generate code with highinfl_df View recommended plots New interactive sheet

```
#view the last few rows and columns for high for retweet
highinfl_df.tail()
```

	A	B
43386	997528560482050048	718863328060289025
43387	997528560482050048	778714123
43388	997528560482050048	872947287584120832
43389	997528560482050048	873135988440223745
43390	997528560482050048	916972497240719361

```
#view the first few rows and columns
lowinfl_df.head()
```

	A	B
0	100256670	1063468160127385601
1	100256670	1070083706822438912
2	100256670	1075392210
3	100256670	1082048954269818880
4	100256670	1171150024983470080

```
#view the last few rows and columns for low for retweet
lowinfl_df.tail()
```

	A	B
52512	997528560482050048	967853319166230529
52513	997528560482050048	986380469959909376
52514	997528560482050048	987148809561141248
52515	997528560482050048	993278238037299200
52516	997528560482050048	998693754981703682

```
#view the first few rows and columns
combined_data.head()
```

	A	B
0	100256670	105327432
1	100256670	1058038885765255175
2	100256670	1064545651
3	100256670	109071031
4	100256670	1120698997327294464

```
#view the last few rows and columns
combined_data.tail()
```

	A	B
88713	997528560482050048	967853319166230529
88714	997528560482050048	986380469959909376
88715	997528560482050048	987148809561141248
88716	997528560482050048	993278238037299200
88717	997528560482050048	998693754981703682

```
highinfl_df['B']=highinfl_df['B'].astype(int)
highinfl_df['A']=highinfl_df['A'].astype(int)
lowinfl_df['B']=lowinfl_df['B'].astype(int)
lowinfl_df['A']=lowinfl_df['A'].astype(int)
combined_data['A']=combined_data['A'].astype(int)
combined_data['B']=combined_data['B'].astype(int)
```

```
#view the name of columns
highinfl_df.columns
lowinfl_df.columns
combined_data.columns
```

```
Index(['A', 'B'], dtype='object')
```

```
#view the information about the combined for retweet
print(combined_data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 88718 entries, 0 to 88717
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   A       88718 non-null  int64
 1   B       88718 non-null  int64
dtypes: int64(2)
memory usage: 1.4 MB
None
```

```
#view the information about the low infl for retweet
print(lowinfl_df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 46292 entries, 0 to 52516
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   A       46292 non-null  int64
 1   B       46292 non-null  int64
dtypes: int64(2)
memory usage: 1.1 MB
None
```

```
#check of ther are in combined data any mising value
print(combined_data.isnull().sum())
```

```
A    0
B    0
dtype: int64
```

```
#check of there are any missing value in highinfl value
print(highinfl_df.isnull().sum())
```

```
A    0
B    0
dtype: int64
```

```
#check of there are any missing value in lowinl value
print(lowinfl_df.isnull().sum())
```

```
A    0
B    0
dtype: int64
```

```
#summarize the data set by calculating some basic statistcs of combined:
print(combined_data.describe().astype(int).to_string())
```

```
                         A                    B
count                88718                88718
mean    380248972446081856   394877655766910080
std     546029463600453824   561304233132552256
min               14362766                  985
25%              148365388            125143408
50%             1164487040           1223575264
75%     939679358536507392   959342442823694336
max    1513724522792636416  1519721637494595584
```

```
#summarize the data set by calculating some basic statistcs of highinfl for retweet:
print(highinfl_df.describe().astype(int).to_string())
```

```
                         A                    B
count                42426                42426
mean    384745755614000128   334506100817180352
std     542731639062526656   528292801808115008
min               14362766                  985
25%              192374137             88869834
50%             1164487040            731943463
75%     939679358536507392   832524310199824384
max    1506670136790589440  1516654448252428288
```

```
#summarize the data set by calculating some basic statistcs of lowinfl for retweet:
print(lowinfl_df.describe().astype(int).to_string())
```

```
                       A                    B
count              46292                46292
mean   376127730704914944   450207379915731392
std    549007993063470336   584493437133760896
min              14362766                  985
25%             135624348            171022949
50%            1343565854           2191380827
75%    950215424981028864  1071977963562825728
max   1513724522792636416  1519721637494595584
```

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

```
highinfl_df = pd.read_csv('/content/drive/My Drive/Colab Notebooks/cs131/retweets_nobots_uniq_highinfluence.NONOUSER.txt', sep='\t', names=column_names)
df = pd.read_csv('/content/drive/My Drive/Colab Notebooks/cs131/retweets_nobots_uniq_lowinfluence.NONOUSER.txt', sep='\t', names=column_names)
```

```
print(df.columns)
```

```
Index(['A', 'B'], dtype='object')
```

Degree centrality

```
#Load the high infl adj list file into a networkx a graph
G = nx.read_edgelist('/content/drive/My Drive/Colab Notebooks/cs131/retweets_nobots_uniq_highinfluence.NONOUSER.txt',delimiter='\t')
```

```
#calculate the degree centratlity for each node in the graph
degree_centrality=nx.degree_centrality(G)
```

```
#find the node with the highest degree centrality
max_node=max(degree_centrality,key=degree_centrality.get)
max_degree_centrality=degree_centrality[max_node]
```

```
#print the highest degree centeraltiy for each node in the graph
print(f'node with the highest degree centrality is {max_node}with a centrality value of{degree_centrality[max_node]:.3f}.')
```

```
node with the highest degree centrality is 42769304with a centrality value of0.089.
```

```
#print the degree centraltiy for each node in the graph
for node, centrality in degree_centrality.items():
    print(f'Node{node}has degree centrality{centrality:.3f}.')
```

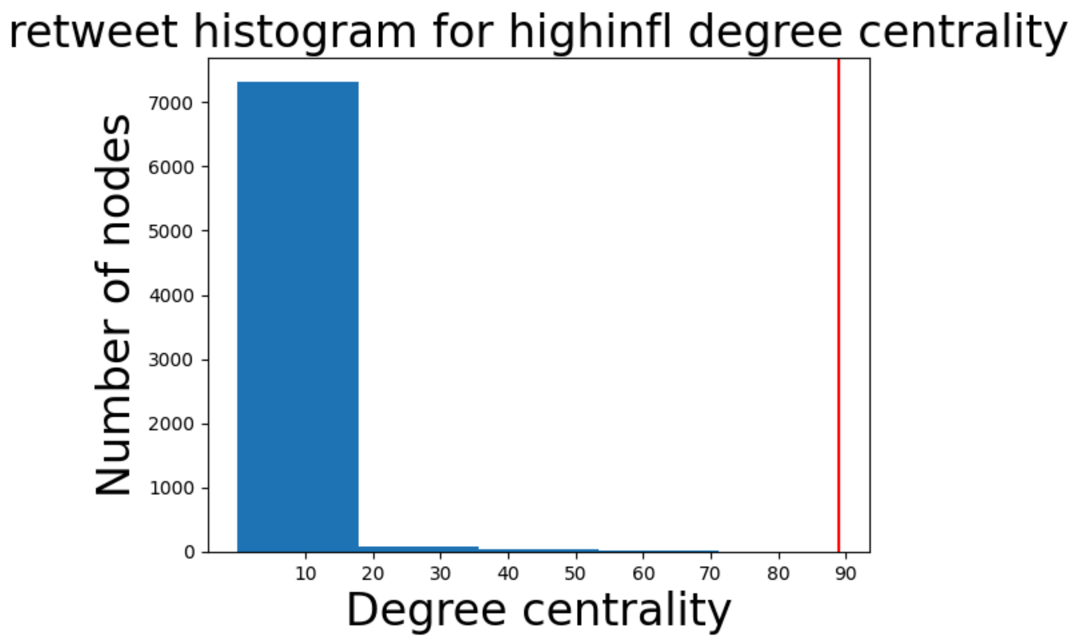
Streaming output truncated to the last 5000 lines.

```
Node1040160799208161280has degree centrality0.001.
Node1042307696has degree centrality0.002.
Node1065677660880338944has degree centrality0.001.
Node1100266332283559936has degree centrality0.000.
Node1123996182has degree centrality0.002.
Node1370616208928747520has degree centrality0.001.
Node1497193035028963365has degree centrality0.001.
Node1542862735has degree centrality0.001.
Node16442365has degree centrality0.000.
Node203307838has degree centrality0.000.
Node22720074has degree centrality0.000.
Node229195516has degree centrality0.001.
Node2315512764has degree centrality0.001.
Node256159524has degree centrality0.001.
Node2600066592has degree centrality0.001.
Node3044584264has degree centrality0.001.
Node311340314has degree centrality0.001.
Node368240745has degree centrality0.002.
Node46817943has degree centrality0.002.
Node4786238905has degree centrality0.000.
Node5900252has degree centrality0.001.
Node717446052698304513has degree centrality0.001.
Node82097756has degree centrality0.001.
Node848691418591891456has degree centrality0.001.
Node949681897has degree centrality0.001.
Node1154228711731990528has degree centrality0.002.
Node1648351has degree centrality0.001.
Node1155192156has degree centrality0.042.
Node1008340728114601984has degree centrality0.001.
Node1080188052365029376has degree centrality0.001.
Node110893874has degree centrality0.001.
Node1120952978393849856has degree centrality0.001.
Node1121967352181030912has degree centrality0.001.
Node113030544has degree centrality0.000.
Node1134152253353447424has degree centrality0.001.
Node1155192538535550976has degree centrality0.002.
Node1175947712124063745has degree centrality0.001.
Node1221554431541633024has degree centrality0.001.
Node1243976208154529795has degree centrality0.001.
Node1244795359097958401has degree centrality0.002.
Node1317940454084825088has degree centrality0.001.
Node1318376979930914816has degree centrality0.000.
Node1321505685813760000has degree centrality0.001.
Node1331107444416770048has degree centrality0.001.
Node1340885244577263616has degree centrality0.000.
Node134758540has degree centrality0.001.
Node1348930324047286272has degree centrality0.001.
Node1378022598752223235has degree centrality0.000.
Node1409613542068613121has degree centrality0.001.
Node1410700164067647495has degree centrality0.001.
Node1425867981112893448has degree centrality0.000.
Node1441585487316062223has degree centrality0.001.
Node1452626837004173313has degree centrality0.000.
Node1465353868624207873has degree centrality0.001.
Node1469151855074762755has degree centrality0.000.
Node1472371170758512653has degree centrality0.000.
Node1479127693077721092has degree centrality0.001.
```

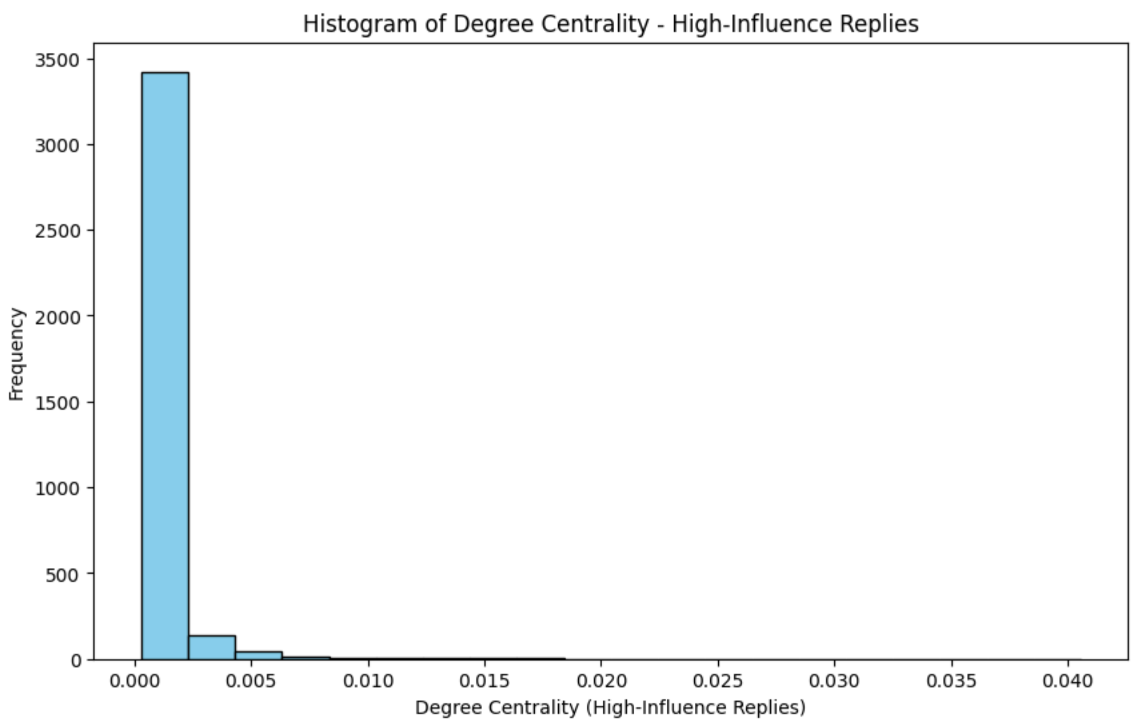
```
#convert the centrality values to integers
degree_centrality={k:int(v*1000)for k,v in degree_centrality.items()}
```

```
#plot a histo gram of the degree centrality values
plt.hist(list(degree_centrality.values()),bins=5)
plt.xlabel('Degree centrality',fontsize=24)
plt.xticks(range(10,100,10))
plt.axvline(x=degree_centrality[max_node],color='red')
plt.ylabel('Number of nodes',fontsize=24)
plt.gca().get_yaxis().get_major_locator().set_params(integer=True)
#plt.yticks([1,2,3,4])
```

```
plt.title('retweet histogram for highinfl degree centrality',fontsize=24)
plt.show()
```




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

# Load the cleaned high-influence replies adjacency list into a NetworkX graph
high_replies_file = '/content/drive/My Drive/Colab Notebooks/cs131/replies_nobots_uniq_highinfluence.NONOUSER.txt'
G_high = nx.read_edgelist(high_replies_file, delimiter='\t', create_using=nx.Graph())

# Calculate Degree Centrality
degree_centrality_high = nx.degree_centrality(G_high)

# Save Degree Centrality to a file
degree_centrality_df_high = pd.DataFrame(list(degree_centrality_high.items()), columns=['Node', 'Degree Centrality'])
degree_centrality_df_high.to_csv('/content/drive/My Drive/Colab Notebooks/cs131/degree_centrality_high_replies.tsv', sep='\t', index=False)

# Plot histogram for Degree Centrality
plt.figure(figsize=(10, 6))
plt.hist(degree_centrality_high.values(), bins=20, color='skyblue', edgecolor='black')
plt.xlabel('Degree Centrality (High-Influence Replies)')
plt.ylabel('Frequency')
plt.title('Histogram of Degree Centrality - High-Influence Replies')
plt.gca().get_yaxis().get_major_locator().set_params(integer=True)
plt.show()
```




```
#Load the high infl adj list file into a networkx a graph
G= nx.read_edgelist('/content/drive/My Drive/Colab Notebooks/cs131/retweets_nobots_uniq_lowinfluence.NONOUSER.txt',delimiter='\t')


#calculate the degree centratlity for each node in the graph
degree_centrality=nx.degree_centrality(G)


#find the node with the highest degree centrality
max_node=max(degree_centrality,key=degree_centrality.get)
max_degree_centrality=degree_centrality[max_node]


print(f'node with the highest degree centrality is {max_node}with a centrality value of{degree_centrality[max_node]:.3f}.')
```

```
node with the highest degree centrality is 2189523500with a centrality value of0.022.
```

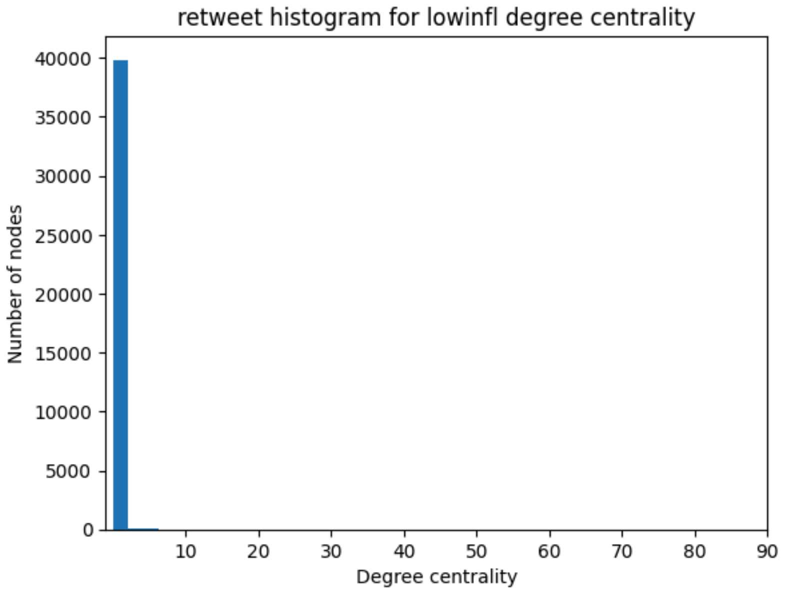
```
#print the degree centraltiy for each node in the graph
for node, centrality in degree_centrality.items():
    print(f'Node{node}has degree centrality{centrality:.3f}.')
```

Streaming output truncated to the last 5000 lines.

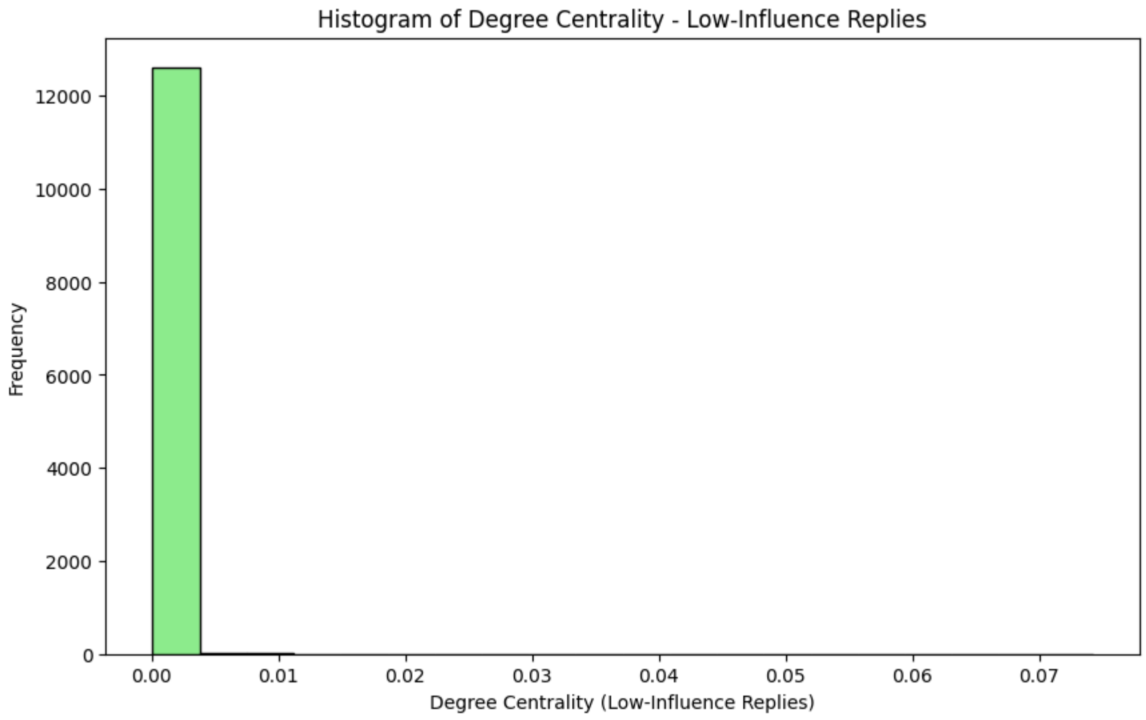
```
Node121594232has degree centrality0.000.
Node1216813913150500866has degree centrality0.000.
Node1221016504067592193has degree centrality0.000.
Node1221377785has degree centrality0.000.
Node1228274850822840320has degree centrality0.000.
Node1231878704743223296has degree centrality0.000.
Node1232653578055147522has degree centrality0.000.
Node1235955986810232833has degree centrality0.000.
Node1241760253has degree centrality0.000.
Node1242142822532292608has degree centrality0.000.
Node1250047514021171201has degree centrality0.000.
Node1251761794974015491has degree centrality0.000.
Node125362283has degree centrality0.000.
```

```
Node1260946135797190656has degree centrality0.000.
Node1280540497has degree centrality0.000.
Node1287118441007259649has degree centrality0.000.
Node128968687has degree centrality0.000.
Node129963753has degree centrality0.000.
Node1304589818has degree centrality0.000.
Node130555328has degree centrality0.000.
Node130640704has degree centrality0.000.
Node1308253814has degree centrality0.000.
Node1315972154517794822has degree centrality0.000.
Node1316398228569546755has degree centrality0.000.
Node1323634389050339328has degree centrality0.000.
Node1324698582495203330has degree centrality0.000.
Node1325265709has degree centrality0.000.
Node1326873186596777985has degree centrality0.000.
Node1328697528825618432has degree centrality0.000.
Node1328699022has degree centrality0.000.
Node1328788482353934340has degree centrality0.000.
Node1334814328084688896has degree centrality0.000.
Node1344651471858630656has degree centrality0.000.
Node1345677728138334208has degree centrality0.000.
Node1346268799has degree centrality0.000.
Node134793154has degree centrality0.000.
Node1355553109959962628has degree centrality0.000.
Node1355828513132187649has degree centrality0.000.
Node1360292790has degree centrality0.000.
Node1363174494299430914has degree centrality0.000.
Node1367853648has degree centrality0.000.
Node1377269138has degree centrality0.000.
Node1378947769184976896has degree centrality0.000.
Node1379292876has degree centrality0.000.
Node1379638549has degree centrality0.000.
Node1381715133706162176has degree centrality0.000.
Node138363206has degree centrality0.000.
Node139230276has degree centrality0.000.
Node1397240360338956292has degree centrality0.000.
Node1398234331135500292has degree centrality0.000.
Node1400580861150613506has degree centrality0.000.
Node1400783012has degree centrality0.000.
Node14048767has degree centrality0.000.
Node14275715has degree centrality0.000.
Node1430446644571607045has degree centrality0.000.
Node1433791346has degree centrality0.000.
Node1439338194has degree centrality0.000.
```

```
#convert the centrality values to integers
degree_centrality={k:int(v*1000)for k,v in degree_centrality.items()}


#plot a histo gram of the degree centrality values
plt.hist(list(degree_centrality.values()),bins=10)
plt.xlabel('Degree centrality')
plt.xticks(range(10,100,10))
#plt.xticks([0,10,20,30,40,50,60,70,80,90,100])
plt.ylabel('Number of nodes')
plt.gca().get_yaxis().get_major_locator().set_params(integer=True)
#plt.yticks([1,2,3,4])
plt.title('retweet histogram for lowinfl degree centrality')
plt.show()
```




```
# Load the cleaned low-influence replies adjacency list into a NetworkX graph
low_replies_file = '/content/drive/My Drive/Colab Notebooks/cs131/replies_nobots_uniq_lowinfluence.NONOUSER.txt'
G_low = nx.read_edgelist(low_replies_file, delimiter='\t', create_using=nx.Graph())

# Calculate Degree Centrality
degree_centrality_low = nx.degree_centrality(G_low)

# Save Degree Centrality to a file
degree_centrality_df_low = pd.DataFrame(list(degree_centrality_low.items()), columns=['Node', 'Degree Centrality'])
degree_centrality_df_low.to_csv('/content/drive/My Drive/Colab Notebooks/cs131/degree_centrality_low_replies.tsv', sep='\t', index=False)

# Plot histogram for Degree Centrality
plt.figure(figsize=(10, 6))
plt.hist(degree_centrality_low.values(), bins=20, color='lightgreen', edgecolor='black')
plt.xlabel('Degree Centrality (Low-Influence Replies)')
plt.ylabel('Frequency')
plt.title('Histogram of Degree Centrality - Low-Influence Replies')
plt.gca().get_yaxis().get_major_locator().set_params(integer=True)
plt.show()
```




```
#Load the high infl adj list file into a networkx a graph
G= nx.read_edgelist('/content/drive/My Drive/Colab Notebooks/cs131/cleaned_combined_AdjList.tsv',delimiter='\t')


#calculate the degree centratlity for each node in the graph
degree_centrality=nx.degree_centrality(G)


#find the node with the highest degree centrality
max_node=max(degree_centrality,key=degree_centrality.get)
max_degree_centrality=degree_centrality[max_node]


print(f'node with the highest degree centrality is {max_node}with a centrality value of{degree_centrality[max_node]:.3f}.')
```

```
node with the highest degree centrality is 42769304with a centrality value of0.030.
```

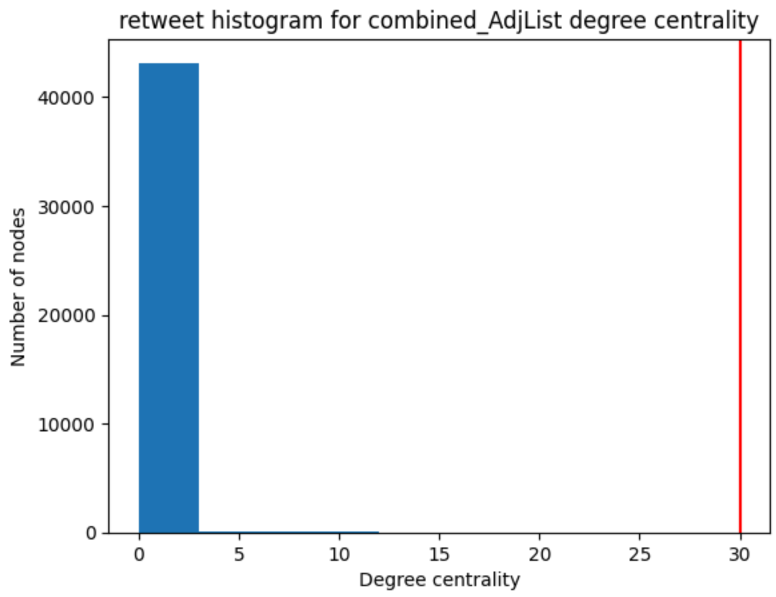
```
#print the degree centraltiy for each node in the graph
for node, centrality in degree_centrality.items():
    print(f'Node{node}has degree centrality{centrality:.3f}.')
```

Streaming output truncated to the last 5000 lines.

```
Node1111094288467800064has degree centrality0.000.
Node1151822814846029824has degree centrality0.000.
Node1194244593170231296has degree centrality0.000.
Node1213953152has degree centrality0.000.
Node1307690532215435264has degree centrality0.000.
Node1330349107has degree centrality0.000.
Node1359952496140156933has degree centrality0.000.
Node1364874243557752833has degree centrality0.000.
Node1387674053389873152has degree centrality0.000.
Node1425101591313530886has degree centrality0.000.
Node1425943814137651212has degree centrality0.000.
Node1440635012has degree centrality0.000.
Node1504202114129530881has degree centrality0.000.
Node1512644840131026949has degree centrality0.000.
Node1525234429has degree centrality0.000.
Node166883389has degree centrality0.000.
Node17092592has degree centrality0.000.
Node19534496has degree centrality0.000.
Node2316064569has degree centrality0.000.
Node23294237has degree centrality0.000.
Node2445637219has degree centrality0.000.
Node263327246has degree centrality0.000.
Node2795748238has degree centrality0.000.
Node291521804has degree centrality0.000.
Node302491448has degree centrality0.000.
Node3060891761has degree centrality0.000.
Node3128782139has degree centrality0.000.
Node31503048has degree centrality0.000.
Node33482561has degree centrality0.000.
Node454696257has degree centrality0.000.
Node4788937961has degree centrality0.000.
Node48575074has degree centrality0.000.
Node485860087has degree centrality0.000.
Node833170111has degree centrality0.000.
Node90996954has degree centrality0.000.
Node975558929584279552has degree centrality0.000.
Node984989302550552576has degree centrality0.000.
Node111400999has degree centrality0.000.
Node1121789470171258880has degree centrality0.000.
Node1274803192203579392has degree centrality0.000.
Node1296035989has degree centrality0.000.
Node1319523522has degree centrality0.000.
Node1326272512808857641has degree centrality0.000.
Node1383377382824837130has degree centrality0.000.
Node1403537423905169412has degree centrality0.000.
Node1403713091972571136has degree centrality0.000.
Node1447742440291504128has degree centrality0.000.
Node1457894861311549444has degree centrality0.000.
Node14730894has degree centrality0.000.
Node177564016has degree centrality0.000.
Node2301990517has degree centrality0.000.
Node242694418has degree centrality0.000.
Node2908170952has degree centrality0.000.
Node394147536has degree centrality0.000.
Node466864852has degree centrality0.000.
Node4765364386has degree centrality0.000.
Node4780832983has degree centrality0.000.
```

```
#convert the centrality values to integers
degree_centrality={k:int(v*1000)for k,v in degree_centrality.items()}


#plot a histo gram of the degree centrality values
plt.hist(list(degree_centrality.values()),bins=10)
plt.xlabel('Degree centrality')
plt.axvline(x=degree_centrality[max_node],color='red')
#plt.xticks(range(0,10,1))
#plt.xticks([0,10,20,30,40,50,60,70,80,90])
plt.ylabel('Number of nodes')
#plt.yticks([5,10,15,20,25,30,35])
plt.gca().get_yaxis().get_major_locator().set_params(integer=True)
```

```
plt.title('retweet histogram for combined_AdjList degree centrality')
plt.show()
```



Betweenness centrality

```
# Create highinfl_df contains the adjacency data
highinfl_df.to_csv('/content/drive/My Drive/Colab Notebooks/cs131/highinfl_AdjList.tsv', sep='\t', index=False, header=False)
```

```
# Create a 1% Random Sample from the Large File
import random

# Input and output file paths
input_file = '/content/drive/My Drive/Colab Notebooks/cs131/highinfl_AdjList.tsv'
output_file = '/content/drive/My Drive/Colab Notebooks/cs131/sampled_highinfl_AdjList.tsv'

# Define sampling rate (1%)
sampling_rate = 0.01

# Create a random sample
with open(input_file, 'r') as infile, open(output_file, 'w') as outfile:
    for line in infile:
        if random.random() < sampling_rate:  # Keep 1% of lines randomly
            outfile.write(line)

print(f"Random sample saved to: {output_file}")
```

```
Random sample saved to: /content/drive/My Drive/Colab Notebooks/cs131/sampled_highinfl_AdjList.tsv
```

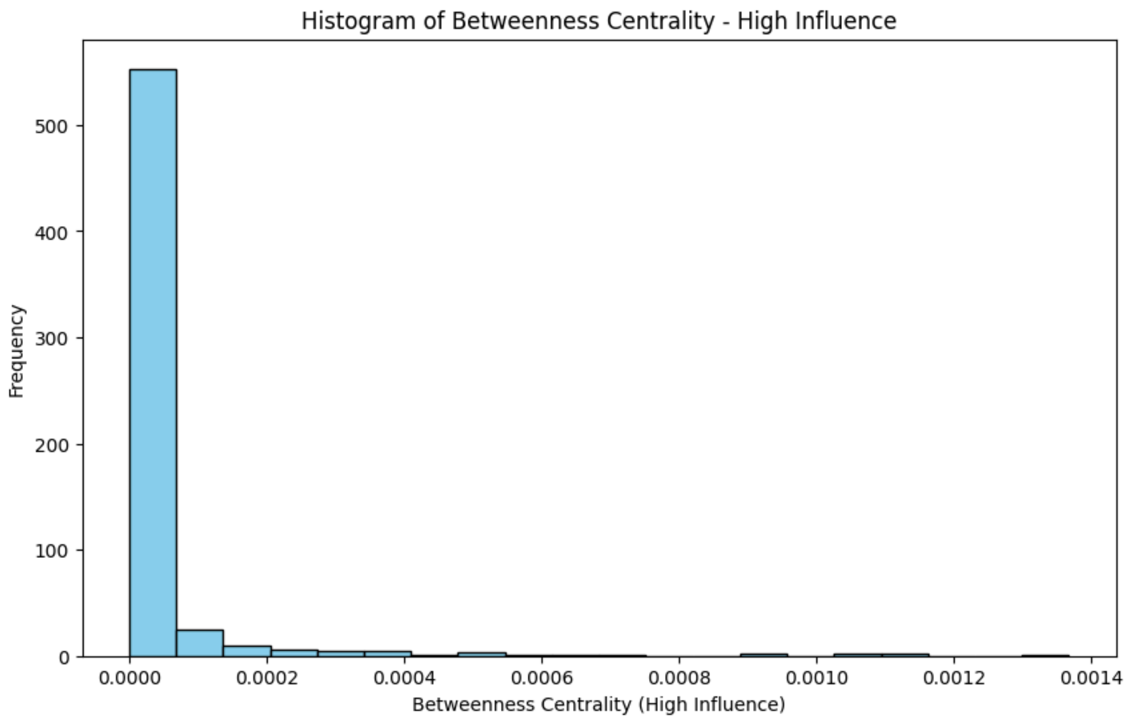
```
# Load the Sampled File into a NetworkX Graph

import networkx as nx
import pandas as pd

# Load the sampled adjacency list
sampled_file = '/content/drive/My Drive/Colab Notebooks/cs131/sampled_highinfl_AdjList.tsv'
G = nx.read_edgelist(sampled_file, delimiter='\t')

# Calculate betweenness centrality
centrality = nx.betweenness_centrality(G)

# Save centrality to a file
centrality_df = pd.DataFrame(list(centrality.items()), columns=['Node', 'Betweenness Centrality'])
centrality_output = '/content/drive/My Drive/Colab Notebooks/cs131/betweenness_centrality_sampled.tsv'
centrality_df.to_csv(centrality_output, sep='\t', index=False)

print(f"Betweenness centrality calculated and saved to: {centrality_output}")
```

```
Betweenness centrality calculated and saved to: /content/drive/My Drive/Colab Notebooks/cs131/betweenness_centrality_sampled.tsv
```

```
#print the betweenness centrality of each node
for node, centrality_score in centrality.items():
    print(f'Node{node}:{centrality_score}')
```

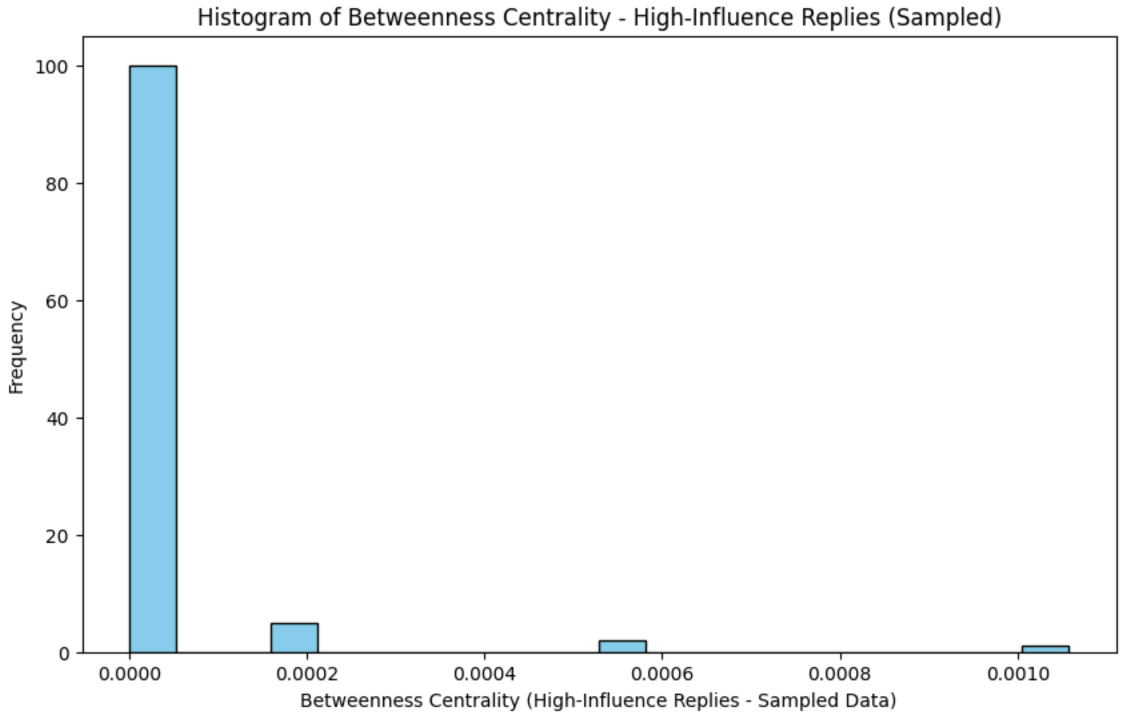
```
Node1291868311:0.0
Node967871677399994368:0.0
Node189868631:0.0
Node968008274003005440:0.0
Node1509220917955682307:0.0
Node971082899381305344:1.738344400792685e-05
Node1507338108:0.0
Node187796215:0.0
Node763010018:0.0
Node977710998:0.0
Node3156867768:0.0
Node981310761862606848:0.00012168410805548795
Node1017551740319797248:0.0
Node14764240:0.0
Node981436419087523840:0.0
Node255812611:0.0
Node982511286:0.0
Node473482276:0.0
Node989200133215617024:0.0
Node1059675376111312897:0.0
Node997262774:0.0
Node1187055258712924162:0.0
```

```
import matplotlib.pyplot as plt

# Plot histogram for betweenness centrality
plt.figure(figsize=(10, 6))
plt.hist(centrality.values(), bins=20, color='skyblue', edgecolor='black')
plt.xlabel('Betweenness Centrality (High Influence)')
plt.ylabel('Frequency')
plt.gca().get_yaxis().get_major_locator().set_params(integer=True)  # Ensure y-axis has integer ticks
plt.title('Histogram of Betweenness Centrality - High Influence')
plt.show()
```




```
import random
import networkx as nx
import pandas as pd
import matplotlib.pyplot as plt

# Input and output file paths for high-influence replies
input_file = '/content/drive/My Drive/Colab Notebooks/cs131/replies_nobots_uniq_highinfluence.NONOUSER.txt'
sampled_file = '/content/drive/My Drive/Colab Notebooks/cs131/sampled_replies_highinfluence.txt'

# Step 1: Create a random 1% sample of the adjacency list
sampling_rate = 0.01
with open(input_file, 'r') as infile, open(sampled_file, 'w') as outfile:
    for line in infile:
        if random.random() < sampling_rate:  # Keep 1% of lines randomly
            outfile.write(line)

print(f"Random sample saved to: {sampled_file}")

# Step 2: Load the sampled adjacency list into a NetworkX graph
G_high = nx.read_edgelist(sampled_file, delimiter='\t', create_using=nx.Graph())

# Step 3: Calculate Betweenness Centrality
betweenness_centrality_high = nx.betweenness_centrality(G_high)

# Step 4: Save Betweenness Centrality to a file
betweenness_df_high = pd.DataFrame(list(betweenness_centrality_high.items()), columns=['Node', 'Betweenness Centrality'])
betweenness_df_high.to_csv('/content/drive/My Drive/Colab Notebooks/cs131/betweenness_centrality_sampled_high_replies.tsv', sep='\t', index=False)

# Step 5: Plot histogram for Betweenness Centrality
plt.figure(figsize=(10, 6))
plt.hist(betweenness_centrality_high.values(), bins=20, color='skyblue', edgecolor='black')
plt.xlabel('Betweenness Centrality (High-Influence Replies - Sampled Data)')
plt.ylabel('Frequency')
plt.gca().get_yaxis().get_major_locator().set_params(integer=True)  # Ensure y-axis has integer ticks
plt.title('Histogram of Betweenness Centrality - High-Influence Replies (Sampled)')
plt.show()
```

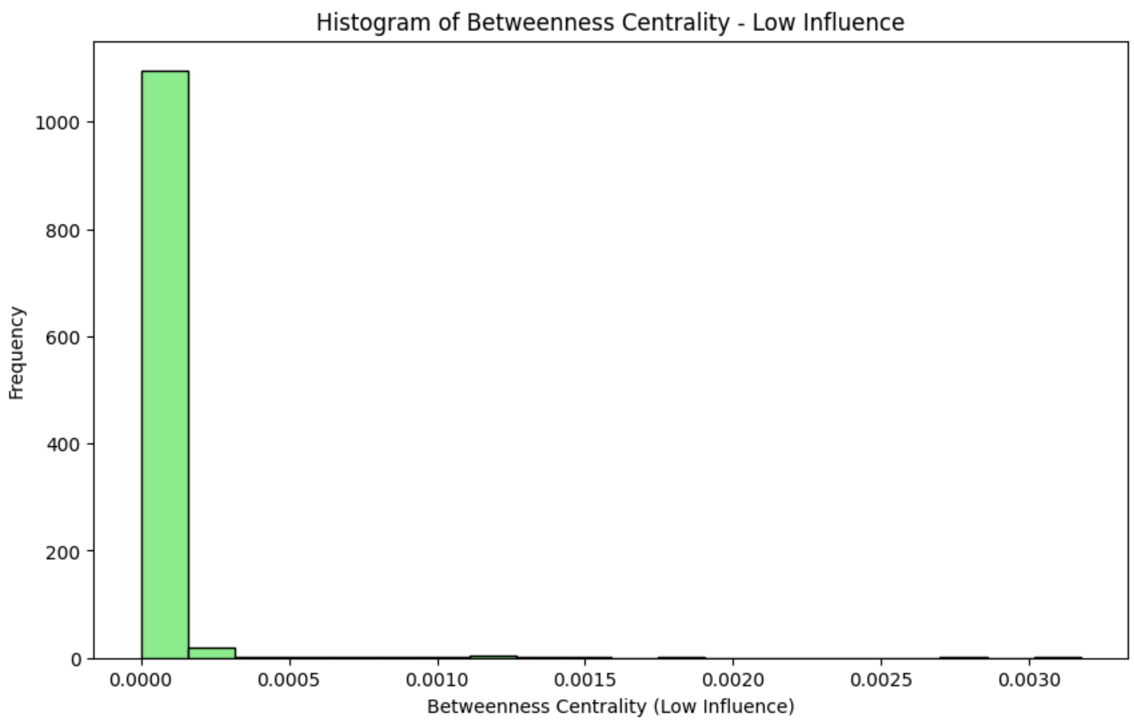
```
Random sample saved to: /content/drive/My Drive/Colab Notebooks/cs131/sampled_replies_highinfluence.txt
```




```
# Create a 1% Random Sample from the Large File
import random

# Input and output file paths
input_file = '/content/drive/My Drive/Colab Notebooks/cs131/lowinfl_AdjList.tsv'
output_file = '/content/drive/My Drive/Colab Notebooks/cs131/sampled_lowinfl_AdjList.tsv'

# Define sampling rate (1%)
sampling_rate = 0.01

# Create a random sample
with open(input_file, 'r') as infile, open(output_file, 'w') as outfile:
    for line in infile:
        if random.random() < sampling_rate:  # Keep 1% of lines randomly
            outfile.write(line)

print(f"Random sample saved to: {output_file}")

# Load the Sampled File into a NetworkX Graph
import networkx as nx
import pandas as pd

# Load the sampled adjacency list
sampled_file = '/content/drive/My Drive/Colab Notebooks/cs131/sampled_lowinfl_AdjList.tsv'
G = nx.read_edgelist(sampled_file, delimiter='\t')

# Calculate betweenness centrality
centrality = nx.betweenness_centrality(G)

# Save centrality to a file
centrality_df = pd.DataFrame(list(centrality.items()), columns=['Node', 'Betweenness Centrality'])
centrality_output = '/content/drive/My Drive/Colab Notebooks/cs131/betweenness_centrality_sampled_low.tsv'
centrality_df.to_csv(centrality_output, sep='\t', index=False)

print(f"Betweenness centrality calculated and saved to: {centrality_output}")
```

```
Random sample saved to: /content/drive/My Drive/Colab Notebooks/cs131/sampled_lowinfl_AdjList.tsv
Betweenness centrality calculated and saved to: /content/drive/My Drive/Colab Notebooks/cs131/betweenness_centrality_sampled_low.tsv
```

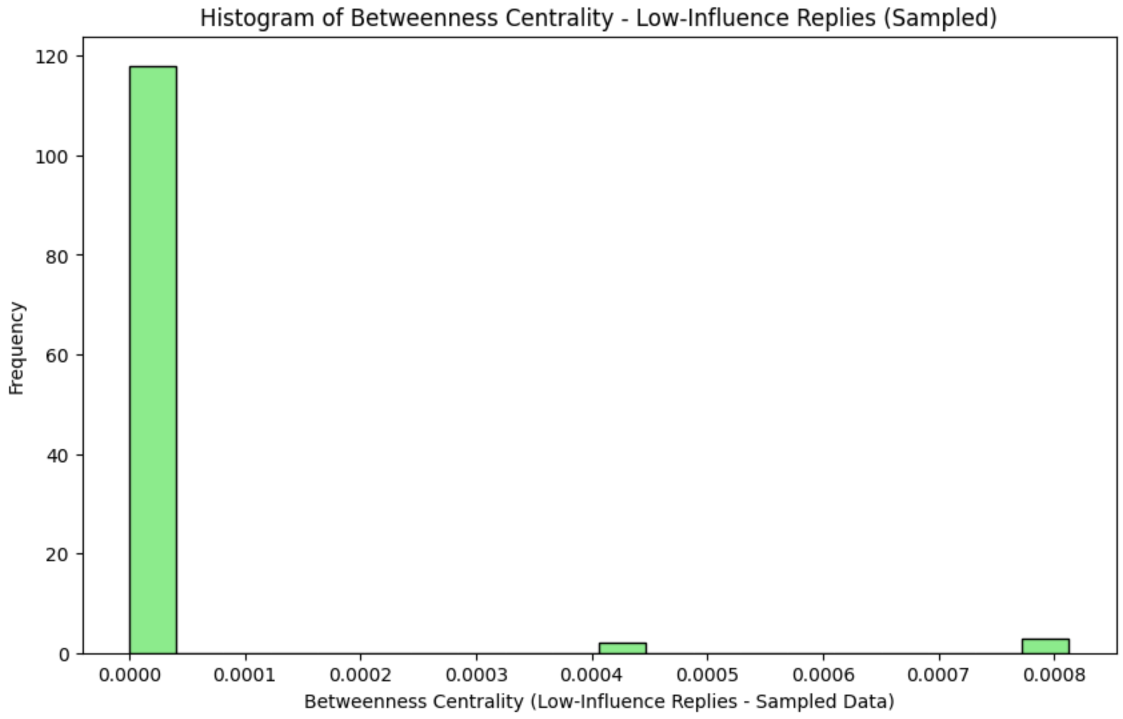
```
import matplotlib.pyplot as plt

# Plot histogram for betweenness centrality
plt.figure(figsize=(10, 6))
plt.hist(centrality.values(), bins=20, color='lightgreen', edgecolor='black')
plt.xlabel('Betweenness Centrality (Low Influence)')
plt.ylabel('Frequency')
plt.gca().get_yaxis().get_major_locator().set_params(integer=True)  # Ensure y-axis has integer ticks
plt.title('Histogram of Betweenness Centrality - Low Influence')
plt.show()
```




```
import random
import networkx as nx
import pandas as pd
import matplotlib.pyplot as plt
```

```
# Input and output file paths for low-influence replies
input_file = '/content/drive/My Drive/Colab Notebooks/cs131/replies_nobots_uniq_lowinfluence.NONOUSER.txt'
sampled_file = '/content/drive/My Drive/Colab Notebooks/cs131/sampled_replies_lowinfluence.txt'

# Step 1: Create a random 1% sample of the adjacency list
sampling_rate = 0.01
with open(input_file, 'r') as infile, open(sampled_file, 'w') as outfile:
    for line in infile:
        if random.random() < sampling_rate:  # Keep 1% of lines randomly
            outfile.write(line)

print(f"Random sample saved to: {sampled_file}")

# Step 2: Load the sampled adjacency list into a NetworkX graph
G_low = nx.read_edgelist(sampled_file, delimiter='\t', create_using=nx.Graph())

# Step 3: Calculate Betweenness Centrality
betweenness_centrality_low = nx.betweenness_centrality(G_low)

# Step 4: Save Betweenness Centrality to a file
betweenness_df_low = pd.DataFrame(list(betweenness_centrality_low.items()), columns=['Node', 'Betweenness Centrality'])
betweenness_df_low.to_csv('/content/drive/My Drive/Colab Notebooks/cs131/betweenness_centrality_sampled_low_replies.tsv', sep='\t', index=False)

# Step 5: Plot histogram for Betweenness Centrality
plt.figure(figsize=(10, 6))
plt.hist(betweenness_centrality_low.values(), bins=20, color='lightgreen', edgecolor='black')
plt.xlabel('Betweenness Centrality (Low-Influence Replies - Sampled Data)')
plt.ylabel('Frequency')
plt.gca().get_yaxis().get_major_locator().set_params(integer=True)  # Ensure y-axis has integer ticks
plt.title('Histogram of Betweenness Centrality - Low-Influence Replies (Sampled)')
plt.show()
```

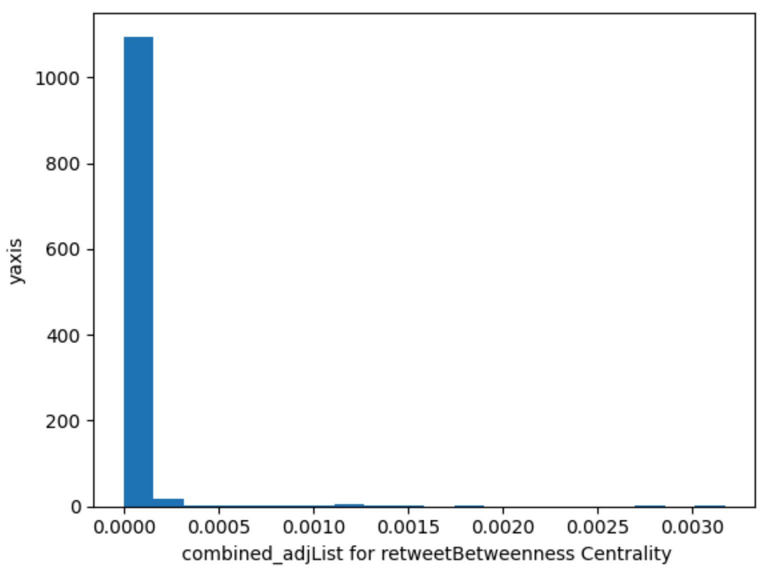
```
Random sample saved to: /content/drive/My Drive/Colab Notebooks/cs131/sampled_replies_lowinfluence.txt
```




```
# Create a 1% Random Sample from the Large File
import random

# Input and output file paths
input_file = '/content/drive/My Drive/Colab Notebooks/cs131/combined_AdjList.tsv'
output_file = '/content/drive/My Drive/Colab Notebooks/cs131/sampled_combined_AdjList.tsv'

# Define sampling rate (1%)
sampling_rate = 0.01

# Create a random sample
with open(input_file, 'r') as infile, open(output_file, 'w') as outfile:
    for line in infile:
        if random.random() < sampling_rate:  # Keep 1% of lines randomly
            outfile.write(line)

print(f"Random sample saved to: {output_file}")

# Load the Sampled File into a NetworkX Graph
import networkx as nx
import pandas as pd

# Load the sampled adjacency list
sampled_file = '/content/drive/My Drive/Colab Notebooks/cs131/sampled_combined_AdjList.tsv'
G = nx.read_edgelist(sampled_file, delimiter='\t')

# Calculate betweenness centrality
centrality = nx.betweenness_centrality(G)

# Save centrality to a file
centrality_df = pd.DataFrame(list(centrality.items()), columns=['Node', 'Betweenness Centrality'])
centrality_output = '/content/drive/My Drive/Colab Notebooks/cs131/betweenness_centrality_sampled_combined.tsv'
centrality_df.to_csv(centrality_output, sep='\t', index=False)

print(f"Betweenness centrality calculated and saved to: {centrality_output}")
```

```
Random sample saved to: /content/drive/My Drive/Colab Notebooks/cs131/sampled_combined_AdjList.tsv
Betweenness centrality calculated and saved to: /content/drive/My Drive/Colab Notebooks/cs131/betweenness_centrality_sampled_combined.tsv
```

```
#plot on histogram
plt.hist(centrality.values(),bins=20)
plt.xlabel(' combined_adjList for retweetBetweenness Centrality')
plt.ylabel('yaxis')
plt.gca().get_yaxis().get_major_locator().set_params(integer=True)
plt.show()
```



Eigenvector centrality

```
#Load the adjcency list from a tsv file
file_path = '/content/drive/My Drive/Colab Notebooks/cs131/highinfl_AdjList.tsv'
```

```
#calculate the betweenness centrality of eACH NODE
eigenvector_centrality=nx.eigenvector_centrality(G,max_iter=1000)
```

```
#find the max node with eigenvector centrality
max_node=max(eigenvector_centrality,key=eigenvector_centrality.get)
```

```
print('Node',max_node,'has the maximum eigenvector centrality of',eigenvector_centrality[max_node])
```

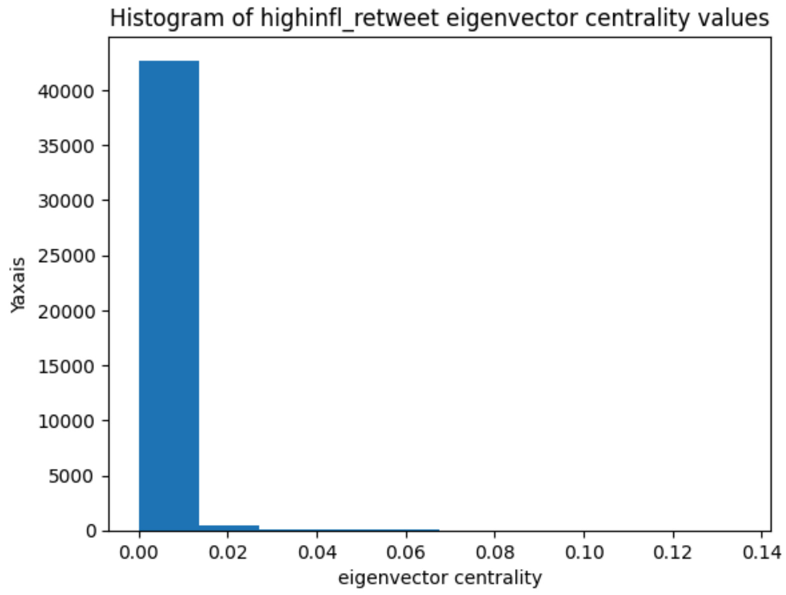
```
Node 723757332 has the maximum eigenvector centrality of 0.13528186100663186
```

```
#print the eigenvector centrality values
for node,ec in eigenvector_centrality.items():
    print(f"{node}:{ec}")
```

Streaming output truncated to the last 5000 lines.

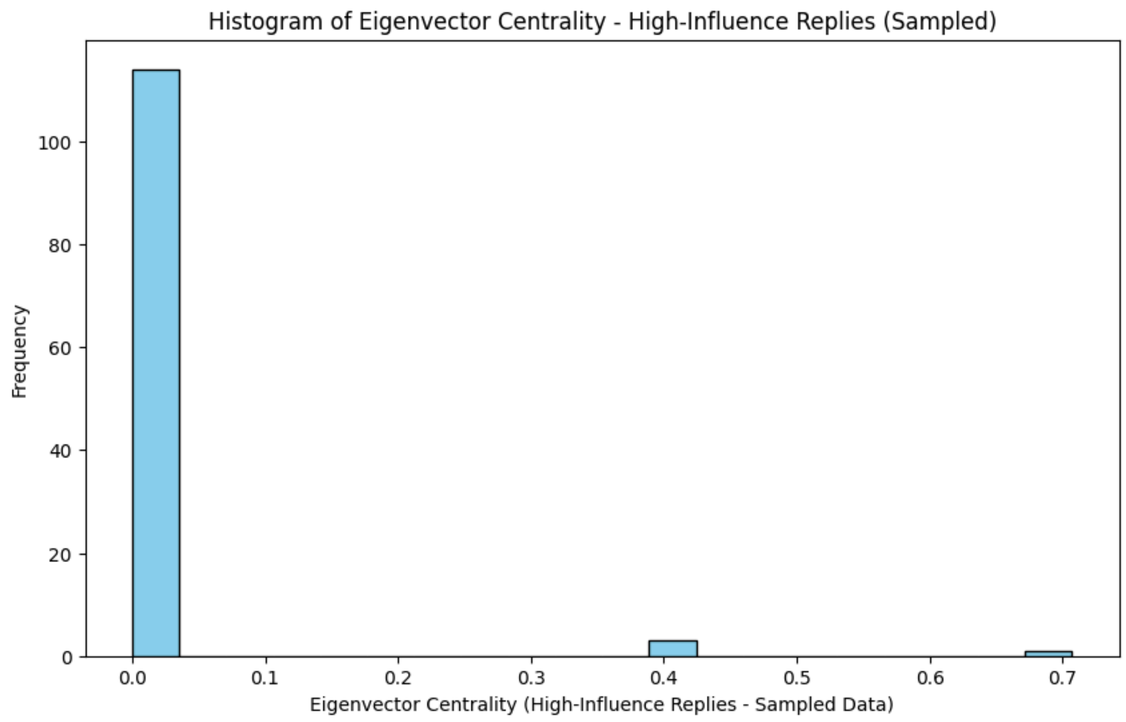
```
1111094288467800064:9.593449465975177e-08
1151822814846029824:9.593449465975177e-08
1194244593170231296:9.593449465975177e-08
1213953152:9.593449465975177e-08
1307690532215435264:9.593449465975177e-08
1330349107:9.593449465975177e-08
1359952496140156933:9.593449465975177e-08
1364874243557752833:9.593449465975177e-08
1387674053389873152:9.593449465975177e-08
1425101591313530886:9.593449465975177e-08
1425943814137651212:9.593449465975177e-08
1440635012:9.593449465975177e-08
1504202114129530881:9.593449465975177e-08
1512644840131026949:9.593449465975177e-08
1525234429:9.593449465975177e-08
166883389:9.593449465975177e-08
17092592:9.593449465975177e-08
19534496:9.593449465975177e-08
2316064569:9.593449465975177e-08
23294237:9.593449465975177e-08
2445637219:9.593449465975177e-08
263327246:9.593449465975177e-08
2795748238:9.593449465975177e-08
291521804:9.593449465975177e-08
302491448:9.593449465975177e-08
3060891761:9.593449465975177e-08
3128782139:9.593449465975177e-08
31503048:9.593449465975177e-08
33482561:9.593449465975177e-08
454696257:9.593449465975177e-08
4788937961:9.593449465975177e-08
48575074:9.593449465975177e-08
485860087:9.593449465975177e-08
833170111:9.593449465975177e-08
90996954:9.593449465975177e-08
975558929584279552:9.593449465975177e-08
984989302550552576:9.593449465975177e-08
111400999:1.3380893514116763e-05
1121789470171258880:1.3380893514116763e-05
1274803192203579392:1.3380893514116763e-05
1296035989:1.3380893514116763e-05
1319523522:1.3380893514116763e-05
1326272512808857641:1.3380893514116763e-05
1383377382824837130:1.3380893514116763e-05
1403537423905169412:1.3380893514116763e-05
1403713091972571136:1.3380893514116763e-05
1447742440291504128:1.3380893514116763e-05
1457894861311549444:1.3380893514116763e-05
14730894:1.3380893514116763e-05
177564016:1.3380893514116763e-05
2301990517:1.3380893514116763e-05
242694418:1.3380893514116763e-05
2908170952:1.3380893514116763e-05
394147536:1.3380893514116763e-05
466864852:1.3380893514116763e-05
4765364386:1.3380893514116763e-05
4780832983:1.3380893514116763e-05
```

```
#plt a histogram of the data
plt.hist(eigenvector_centrality.values(), bins=10)
#Add labels and a title to the plot
plt.xlabel('eigenvector centrality')
plt.ylabel('Yaxais')
plt.gca().get_yaxis().get_major_locator().set_params(integer=True)
plt.title('Histogram of highinfl_retweet eigenvector centrality values')
#show the plot
plt.show()
```




```
import random
import networkx as nx
import pandas as pd
import matplotlib.pyplot as plt

# Input and output file paths for high-influence replies
input_file = '/content/drive/My Drive/Colab Notebooks/cs131/replies_nobots_uniq_highinfluence.NONOUSER.txt'
sampled_file = '/content/drive/My Drive/Colab Notebooks/cs131/sampled_replies_highinfluence.txt'

# Step 1: Create a random 1% sample of the adjacency list
sampling_rate = 0.01
with open(input_file, 'r') as infile, open(sampled_file, 'w') as outfile:
    for line in infile:
        if random.random() < sampling_rate:  # Keep 1% of lines randomly
            outfile.write(line)

print(f"Random sample saved to: {sampled_file}")

# Step 2: Load the sampled adjacency list into a NetworkX graph
G_high = nx.read_edgelist(sampled_file, delimiter='\t', create_using=nx.Graph())

# Step 3: Calculate Eigenvector Centrality
eigenvector_centrality_high = nx.eigenvector_centrality(G_high, max_iter=1000)

# Step 4: Save Eigenvector Centrality to a file
eigenvector_df_high = pd.DataFrame(list(eigenvector_centrality_high.items()), columns=['Node', 'Eigenvector Centrality'])
eigenvector_df_high.to_csv('/content/drive/My Drive/Colab Notebooks/cs131/eigenvector_centrality_sampled_high_replies.tsv', sep='\t', index=False)

# Step 5: Plot histogram for Eigenvector Centrality
plt.figure(figsize=(10, 6))
plt.hist(eigenvector_centrality_high.values(), bins=20, color='skyblue', edgecolor='black')
plt.xlabel('Eigenvector Centrality (High-Influence Replies - Sampled Data)')
plt.ylabel('Frequency')
plt.gca().get_yaxis().get_major_locator().set_params(integer=True)  # Ensure y-axis has integer ticks
plt.title('Histogram of Eigenvector Centrality - High-Influence Replies (Sampled)')
plt.show()
```

```
Random sample saved to: /content/drive/My Drive/Colab Notebooks/cs131/sampled_replies_highinfluence.txt
```



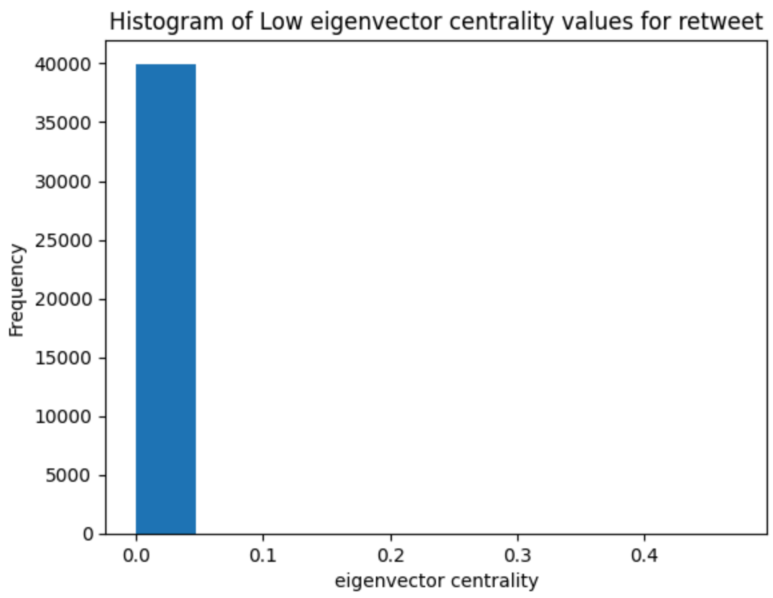

```
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np


file_path = '/content/drive/My Drive/Colab Notebooks/cs131/lowinfl_AdjList.tsv'

# Load the adjacency list into a NetworkX graph
G = nx.read_edgelist(file_path, delimiter='\t')


#calculate the betweenness centrality of eACH NODE
eig_cen=nx.eigenvector_centrality(G,max_iter=1000)


#find the max node with eigenvector centrality
max_node=max(eig_cen,key=eig_cen.get)


print('Node',max_node,'has the maximum eigenvector centrality of',eig_cen[max_node])
```

```
Node 2189523500 has the maximum eigenvector centrality of 0.47240487766026906
```

```
#print the eigenvector centrality values
for node,ec in eig_cen.items():
    print(f"{node}:{ec}")
```

```
Streaming output truncated to the last 5000 lines.
121594232:4.739629457431875e-07
1216813913150500866:4.739629457431875e-07
1221016504067592193:4.739629457431875e-07
1221377785:4.739629457431875e-07
1228274850822840320:4.739629457431875e-07
1231878704743223296:4.739629457431875e-07
1232653578055147522:4.739629457431875e-07
1235955986810232833:4.739629457431875e-07
1241760253:4.739629457431875e-07
1242142822532292608:4.739629457431875e-07
1250047514021171201:4.739629457431875e-07
1251761794974015491:4.739629457431875e-07
125362283:4.739629457431875e-07
1260946135797190656:4.739629457431875e-07
1280540497:4.560952171757735e-06
1287118441007259649:4.739629457431875e-07
128968687:4.739629457431875e-07
129963753:4.739629457431875e-07
1304589818:4.739629457431875e-07
130555328:4.739629457431875e-07
130640704:4.739629457431875e-07
1308253814:4.739629457431875e-07
1315972154517794822:4.739629457431875e-07
1316398228569546755:4.739629457431875e-07
1323634389050339328:4.739629457431875e-07
1324698582495203330:4.739629457431875e-07
1325265709:4.739629457431875e-07
1326873186596777985:4.739629457431875e-07
1328697528825618432:4.739629457431875e-07
1328699022:4.739629457431875e-07
1328788482353934340:4.739629457431875e-07
1334814328084688896:4.739629457431875e-07
1344651471858630656:4.739629457431875e-07
1345677728138334208:4.739629457431875e-07
1346268799:4.739629457431875e-07
134793154:4.739629457431875e-07
1355553109959962628:4.739629457431875e-07
1355828513132187649:4.739629457431875e-07
1360292790:4.739629457431875e-07
1363174494299430914:4.739629457431875e-07
1367853648:4.739629457431875e-07
1377269138:4.739629457431875e-07
1378947769184976896:4.739629457431875e-07
1379292876:4.739629457431875e-07
1379638549:4.739629457431875e-07
1381715133706162176:4.739629457431875e-07
138363206:4.739629457431875e-07
139230276:4.739629457431875e-07
1397240360338956292:4.739629457431875e-07
1398234331135500292:4.739629457431875e-07
1400580861150613506:4.739629457431875e-07
1400783012:4.739629457431875e-07
14048767:4.739629457431875e-07
14275715:4.739629457431875e-07
1430446644571607045:4.739629457431875e-07
1433791346:4.739629457431875e-07
1439338194:4.739629457431875e-07
```

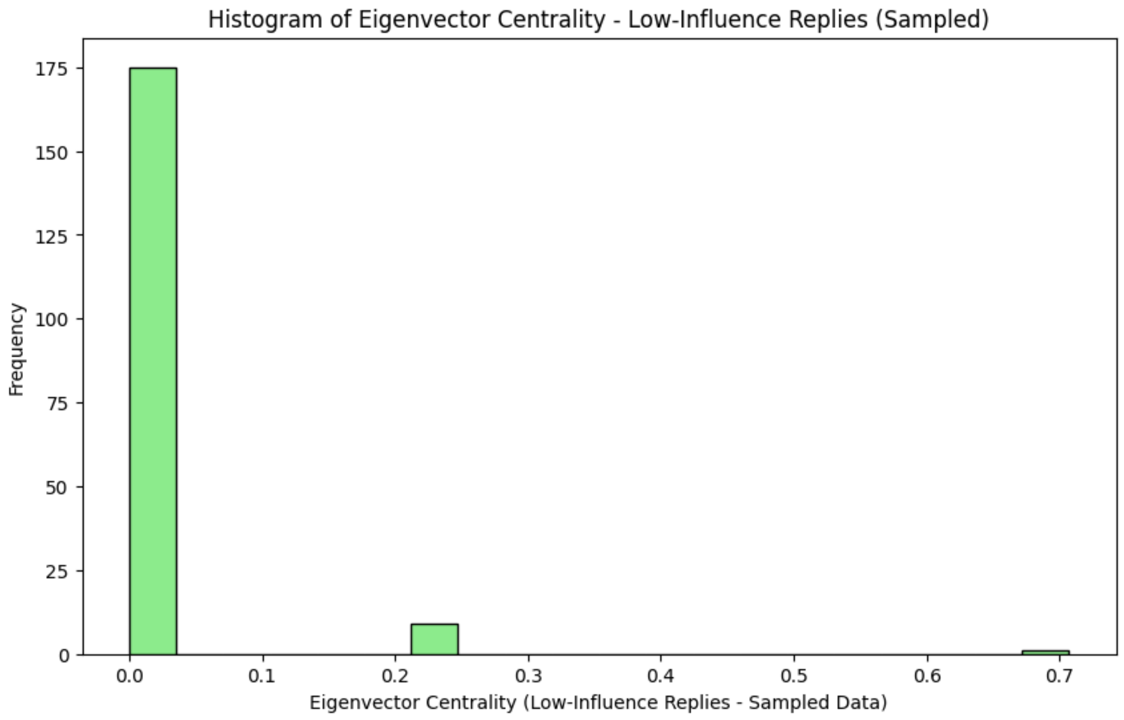
```
#plt a histogram of the data
plt.hist(eig_cen.values(), bins=10)
#Add labels and a title to the plot
plt.xlabel('eigenvector centrality')
plt.ylabel('Frequency')
plt.gca().get_yaxis().get_major_locator().set_params(integer=True)
plt.title('Histogram of Low eigenvector centrality values for retweet')
#show the plot
plt.show()
```




```
# Input and output file paths for low-influence replies
input_file = '/content/drive/My Drive/Colab Notebooks/cs131/replies_nobots_uniq_lowinfluence.NONOUSER.txt'
sampled_file = '/content/drive/My Drive/Colab Notebooks/cs131/sampled_replies_lowinfluence.txt'

# Step 1: Create a random 1% sample of the adjacency list
sampling_rate = 0.01
with open(input_file, 'r') as infile, open(sampled_file, 'w') as outfile:
    for line in infile:
        if random.random() < sampling_rate:  # Keep 1% of lines randomly
            outfile.write(line)

print(f"Random sample saved to: {sampled_file}")

# Step 2: Load the sampled adjacency list into a NetworkX graph
G_low = nx.read_edgelist(sampled_file, delimiter='\t', create_using=nx.Graph())

# Step 3: Calculate Eigenvector Centrality
eigenvector_centrality_low = nx.eigenvector_centrality(G_low, max_iter=1000)

# Step 4: Save Eigenvector Centrality to a file
eigenvector_df_low = pd.DataFrame(list(eigenvector_centrality_low.items()), columns=['Node', 'Eigenvector Centrality'])
eigenvector_df_low.to_csv('/content/drive/My Drive/Colab Notebooks/cs131/eigenvector_centrality_sampled_low_replies.tsv', sep='\t', index=False)

# Step 5: Plot histogram for Eigenvector Centrality
plt.figure(figsize=(10, 6))
plt.hist(eigenvector_centrality_low.values(), bins=20, color='lightgreen', edgecolor='black')
```

```
plt.hist(eigenvector_centrality_low.values(), bins=20, color='lightgreen', edgecolor='black')
plt.xlabel('Eigenvector Centrality (Low-Influence Replies - Sampled Data)')
plt.ylabel('Frequency')
plt.gca().get_yaxis().get_major_locator().set_params(integer=True)  # Ensure y-axis has integer ticks
plt.title('Histogram of Eigenvector Centrality - Low-Influence Replies (Sampled)')
plt.show()
```

```
Random sample saved to: /content/drive/My Drive/Colab Notebooks/cs131/sampled_replies_lowinfluence.txt
```




```
# Save combined_data as combined_AdjList.tsv
combined_data.to_csv('/content/drive/My Drive/Colab Notebooks/cs131/combined_AdjList.tsv', sep='\t', index=False, header=False)

#print("File combined_AdjList.tsv has been created successfully.")
```

```
file_path = '/content/drive/My Drive/Colab Notebooks/cs131/combined_AdjList.tsv'

G = nx.read_edgelist(file_path, delimiter='\t')
```

```
#calculate the betweenness centrality of eACH NODE
eig_cen=nx.eigenvector_centrality(G,max_iter=1000)


#find the max node with eigenvector centrality
max_node=max(eig_cen,key=eig_cen.get)


print('Node',max_node,'has the maximum eigenvector centrality of',eig_cen[max_node])
```

```
Node 723757332 has the maximum eigenvector centrality of 0.1352818610066319
```

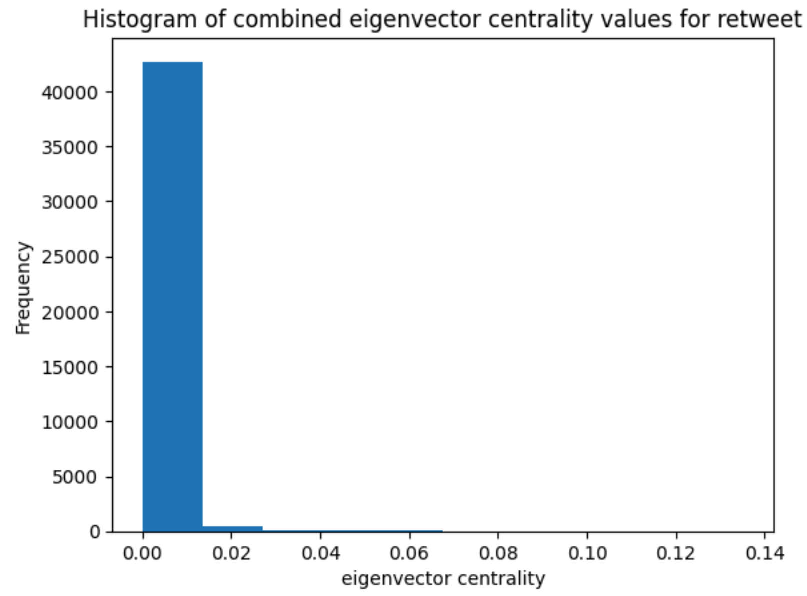
```
#print the eigenvector centrality values
for node,ec in eig_cen.items():
    print(f"{node}:{ec}")
```

Streaming output truncated to the last 5000 lines.

```
1111094288467800064:9.593449465975177e-08
1151822814846029824:9.593449465975177e-08
1194244593170231296:9.593449465975177e-08
1213953152:9.593449465975177e-08
1307690532215435264:9.593449465975177e-08
1330349107:9.593449465975177e-08
1359952496140156933:9.593449465975177e-08
1364874243557752833:9.593449465975177e-08
1387674053389873152:9.593449465975177e-08
1425101591313530886:9.593449465975177e-08
1425943814137651212:9.593449465975177e-08
1440635012:9.593449465975177e-08
1504202114129530881:9.593449465975177e-08
1512644840131026949:9.593449465975177e-08
1525234429:9.593449465975177e-08
166883389:9.593449465975177e-08
17092592:9.593449465975177e-08
19534496:9.593449465975177e-08
2316064569:9.593449465975177e-08
23294237:9.593449465975177e-08
2445637219:9.593449465975177e-08
263327246:9.593449465975177e-08
2795748238:9.593449465975177e-08
291521804:9.593449465975177e-08
302491448:9.593449465975177e-08
3060891761:9.593449465975177e-08
3128782139:9.593449465975177e-08
31503048:9.593449465975177e-08
33482561:9.593449465975177e-08
454696257:9.593449465975177e-08
4788937961:9.593449465975177e-08
48575074:9.593449465975177e-08
485860087:9.593449465975177e-08
833170111:9.593449465975177e-08
90996954:9.593449465975177e-08
975558929584279552:9.593449465975177e-08
984989302550552576:9.593449465975177e-08
111400999:1.3380893514116764e-05
1121789470171258880:1.3380893514116764e-05
1274803192203579392:1.3380893514116764e-05
1296035989:1.3380893514116764e-05
1319523522:1.3380893514116764e-05
1326272512808857641:1.3380893514116764e-05
1383377382824837130:1.3380893514116764e-05
1403537423905169412:1.3380893514116764e-05
1403713091972571136:1.3380893514116764e-05
1447742440291504128:1.3380893514116764e-05
1457894861311549444:1.3380893514116764e-05
14730894:1.3380893514116764e-05
177564016:1.3380893514116764e-05
2301990517:1.3380893514116764e-05
242694418:1.3380893514116764e-05
2908170952:1.3380893514116764e-05
```

```
394147536:1.3380893514116764e-05
466864852:1.3380893514116764e-05
4765364386:1.3380893514116764e-05
4780832983:1.3380893514116764e-05
```

```
#plt a histogram of the data
plt.hist(eig_cen.values(), bins=10)
#Add labels and a title to the plot
plt.xlabel('eigenvector centrality')
plt.ylabel('Frequency')
plt.gca().get_yaxis().get_major_locator().set_params(integer=True)
plt.title('Histogram of combined eigenvector centrality values for retweet')
#show the plot
plt.show()
```



K-core centrality

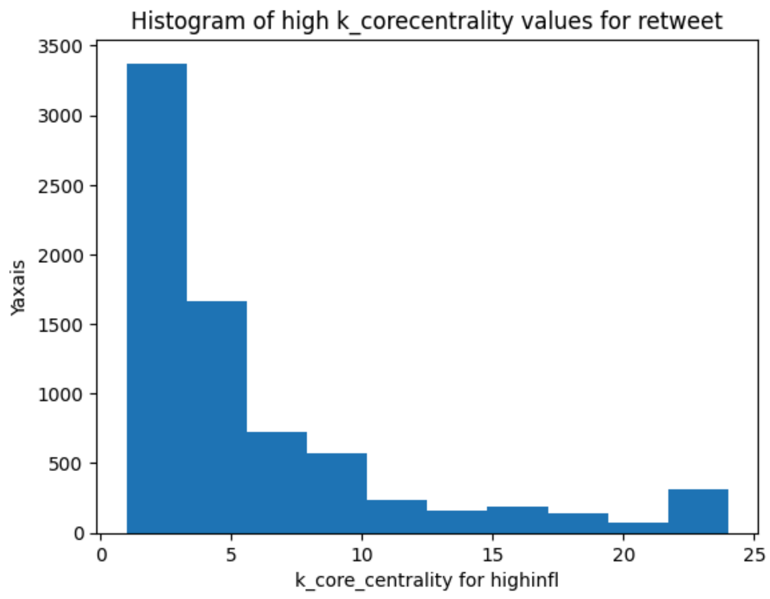
```
import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt
```

```
import networkx as nx

file_path = '/content/drive/My Drive/Colab Notebooks/cs131/retweets_nobots_uniq_highinfluence.NONOUSER.txt'

# Load the adjacency list
G = nx.read_edgelist(file_path, delimiter='\t')


#calculate K-core decomposition
k_core_centralities=nx.core_number(G)


#find the max node for k-core centralities for high
max_k_core_centralities=[node for node,k_core_centrality in k_core_centralities.items()if k_core_centrality==max(k_core_centralities.values())]


#print the nodes with maximum k_core centrality for high
print("The nodes with the maximum k_core centrality value are")
for node in max_k_core_centralities:
    print(node)
```

```
The nodes with the maximum k_core centrality value are
105327432
1205226529455632385
163018653
16563015
1891490382
299273962
30844417
3129968261
380648579
586291040
67934675
720139699
879147821915615233
985749294
1010308883896774656
1120633726478823425
1168793622248067072
118459189
1219232377605644289
1238583088998830081
1296279134079823872
1314241954058833926
14106476
143427448
18831926
2260170266
2420267570
2561718665
2573480784
2749315621
27493883
2758100418
27831488
279390084
3003886063
304679484
33727663
3972812140
469509327
832524310199824384
873135988440223745
87358629
939486738543796224
1252988129708929024
15480566
953924228306305024
1495480590572961792
1640929196
18869484
2288308578
68964628
739844197935644672
```

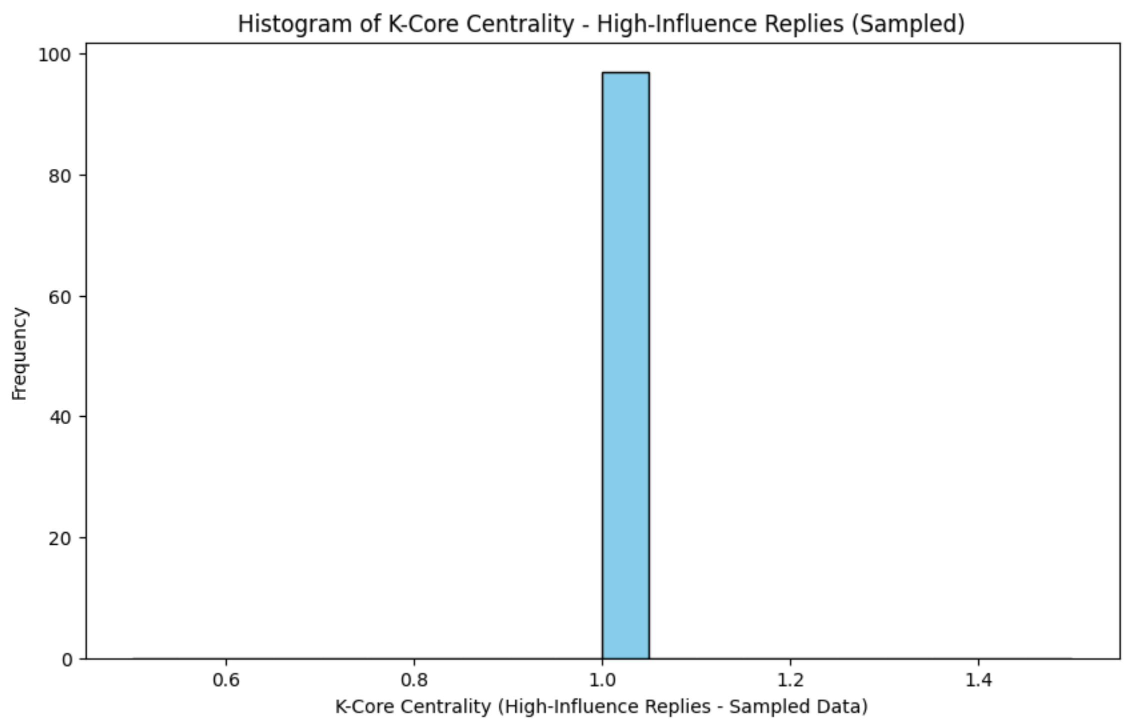
```
1028022611324747776
1003107003693137921
1005846500583321601
1045110787
1243560408025198593
```

```
#plot histogram of k-Core centrality
#plt a histogram of the data
plt.hist(list(k_core_centralities.values()))
#Add labels and a title to the plot
plt.xlabel('k_core_centrality for highinfl')
plt.ylabel('Yaxais')
plt.gca().get_yaxis().get_major_locator().set_params(integer=True)
plt.title('Histogram of high k_corecentrality values for retweet')
#show the plot
plt.show()
```




```
import random
import networkx as nx
import pandas as pd
import matplotlib.pyplot as plt

# Input and output file paths for high-influence replies
input_file = '/content/drive/My Drive/Colab Notebooks/cs131/replies_nobots_uniq_highinfluence.NONOUSER.txt'
sampled_file = '/content/drive/My Drive/Colab Notebooks/cs131/sampled_replies_highinfluence.txt'

# Step 1: Create a random 1% sample of the adjacency list
sampling_rate = 0.01
with open(input_file, 'r') as infile, open(sampled_file, 'w') as outfile:
    for line in infile:
        if random.random() < sampling_rate:  # Keep 1% of lines randomly
            outfile.write(line)

print(f"Random sample saved to: {sampled_file}")

# Step 2: Load the sampled adjacency list into a NetworkX graph
G_high = nx.read_edgelist(sampled_file, delimiter='\t', create_using=nx.Graph())

# Step 3: Calculate K-Core Centrality (coreness)
k_core_high = nx.core_number(G_high)

# Step 4: Save K-Core Centrality to a file
k_core_df_high = pd.DataFrame(list(k_core_high.items()), columns=['Node', 'K-Core'])
k_core_df_high.to_csv('/content/drive/My Drive/Colab Notebooks/cs131/k_core_centrality_sampled_high_replies.tsv', sep='\t', index=False)

# Step 5: Plot histogram for K-Core Centrality
plt.figure(figsize=(10, 6))
plt.hist(k_core_high.values(), bins=20, color='skyblue', edgecolor='black')
plt.xlabel('K-Core Centrality (High-Influence Replies - Sampled Data)')
plt.ylabel('Frequency')
plt.title('Histogram of K-Core Centrality - High-Influence Replies (Sampled)')
plt.gca().get_yaxis().get_major_locator().set_params(integer=True)  # Ensure y-axis has integer ticks
plt.show()
```

```
Random sample saved to: /content/drive/My Drive/Colab Notebooks/cs131/sampled_replies_highinfluence.txt
```




```
import networkx as nx

file_path = '/content/drive/My Drive/Colab Notebooks/cs131/retweets_nobots_uniq_lowinfluence.NONOUSER.txt'

# Load the adjacency list
G = nx.read_edgelist(file_path, delimiter='\t')
```

```
#calculate K-core decomposition
k_core_centralities=nx.core_number(G)


#find the max node for k-core centralities for high
max_k_core_centralities=[node for node,k_core_centrality in k_core_centralities.items()if k_core_centrality==max(k_core_centralities.values())]


#print the nodes with maximum k_core centrality for high
print("The nodes with the maximum k_core centrality value are")
for node in max_k_core_centralities:
    print(node)
```

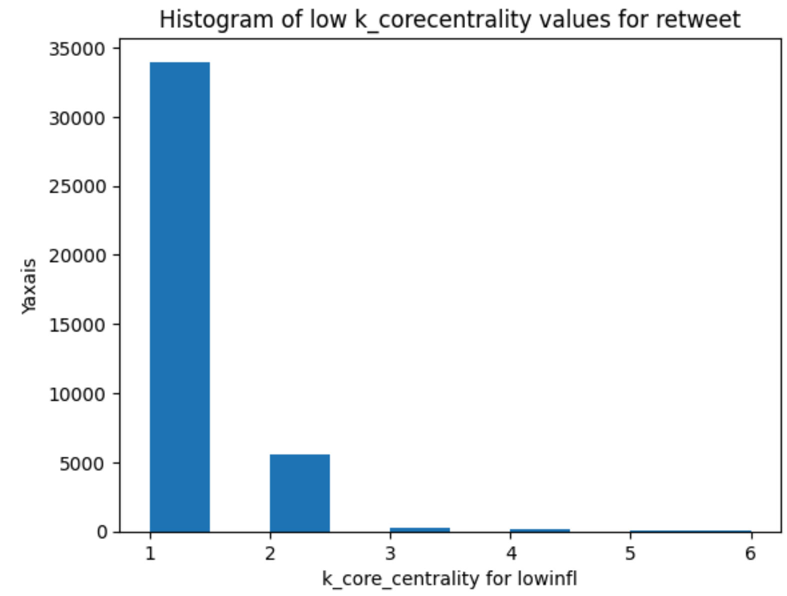
```
The nodes with the maximum k_core centrality value are
2189523500
1016943828052336640
1238370241169616897
132731802
1382912376597479427
1573800344
173672963
18965916
198296897
209535365
2479226604
258124400
29420301
352754946
48624704
588732151
58974496
66917778
711750460932141056
764657028
766136186268160000
79402753
870629200348274688
926019424154349568
2574271742
559681926
629280862
518334117
551483643
```

```
#print the eigenvector centrality values
for node in k_core_centralities:
    print(node)
```

Streaming output truncated to the last 5000 lines.

```
121594232
1216813913150500866
1221016504067592193
1221377785
1228274850822840320
1231878704743223296
1232653578055147522
1235955986810232833
1241760253
1242142822532292608
1250047514021171201
1251761794974015491
125362283
1260946135797190656
1280540497
1287118441007259649
128968687
129963753
1304589818
130555328
130640704
1308253814
1315972154517794822
1316398228569546755
1323634389050339328
1324698582495203330
1325265709
1326873186596777985
1328697528825618432
1328699022
1328788482353934340
1334814328084688896
1344651471858630656
1345677728138334208
1346268799
134793154
1355553109959962628
1355828513132187649
1360292790
1363174494299430914
1367853648
1377269138
1378947769184976896
1379292876
1379638549
1381715133706162176
138363206
139230276
1397240360338956292
1398234331135500292
1400580861150613506
1400783012
14048767
14275715
1430446644571607045
1433791346
1439338194
```

```
#plot histogram of k-Core centrality
#plt a histogram of the data
plt.hist(list(k_core_centralities.values()))
#Add labels and a title to the plot
plt.xlabel('k_core_centrality for lowinfl')
plt.ylabel('Yaxais')
plt.gca().get_yaxis().get_major_locator().set_params(integer=True)
plt.title('Histogram of low k_corecentrality values for retweet')
#show the plot
plt.show()
```




```
import networkx as nx

file_path = '/content/drive/My Drive/Colab Notebooks/cs131/retweets_nobots_uniq_lowinfluence.NONOUSER.txt'

# Load the adjacency list
G = nx.read_edgelist(file_path, delimiter='\t')


#calculate K-core decomposition
k_core_centralities=nx.core_number(G)


#find the max node for k-core centralities for high
max_k_core_centralities=[node for node,k_core_centrality in k_core_centralities.items()if k_core_centrality==max(k_core_centralities.values())]


#print the nodes with maximum k_core centrality for high
print("The nodes with the maximum k_core centrality value are")
for node in max_k_core_centralities:
    print(node)
```

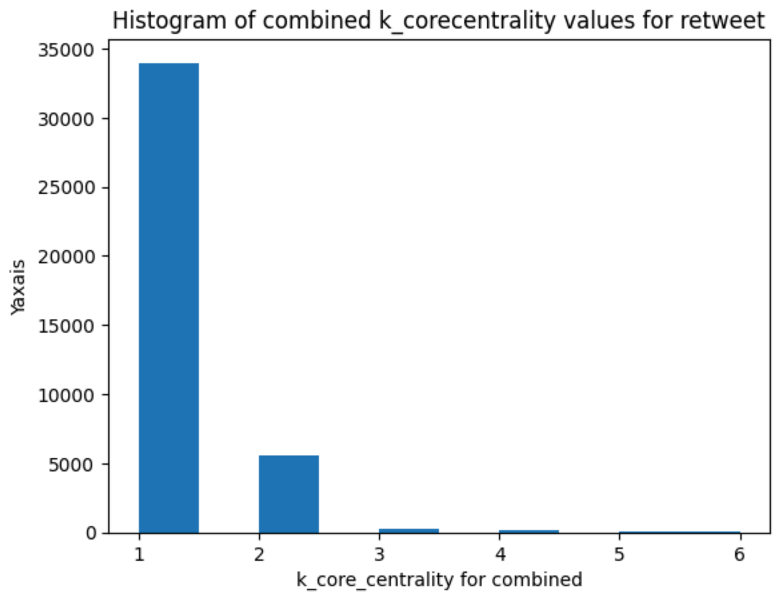
```
The nodes with the maximum k_core centrality value are
2189523500
1016943828052336640
1238370241169616897
132731802
1382912376597479427
1573800344
173672963
18965916
198296897
209535365
2479226604
258124400
29420301
352754946
48624704
588732151
58974496
66917778
711750460932141056
764657028
766136186268160000
79402753
870629200348274688
926019424154349568
2574271742
559681926
629280862
518334117
551483643
```

```
#print the eigenvector centrality values
for node in k_core_centralities:
    print(node)
```

```
Streaming output truncated to the last 5000 lines.
121594232
1216813913150500866
1221016504067592193
1221377785
1228274850822840320
1231878704743223296
1232653578055147522
1235955986810232833
1241760253
1242142822532292608
1250047514021171201
1251761794974015491
125362283
1260946135797190656
1280540497
1287118441007259649
128968687
129963753
1304589818
130555328
130640704
1308253814
1315972154517794822
1316398228569546755
1323634389050339328
1324698582495203330
1325265709
1326873186596777985
1328697528825618432
1328699022
1328788482353934340
1334814328084688896
1344651471858630656
1345677728138334208
1346268799
134793154
1355553109959962628
1355828513132187649
1360292790
1363174494299430914
1367853648
1377269138
1378947769184976896
```

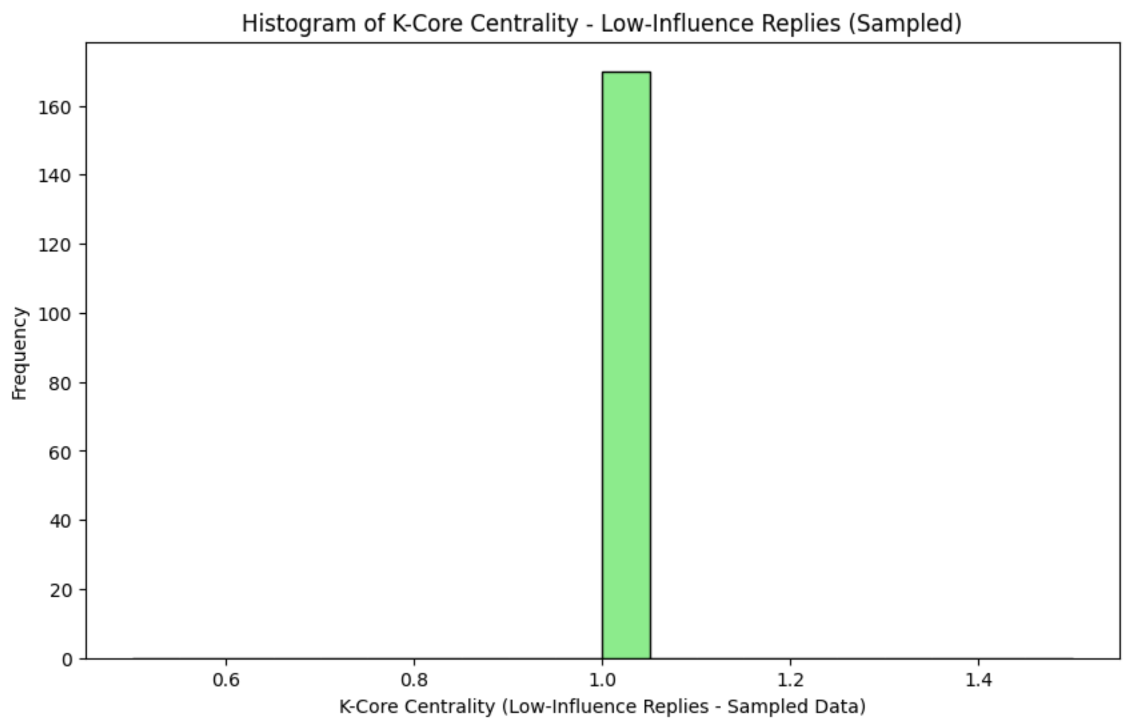
```
1379292876
1379638549
1381715133706162176
138363206
139230276
1397240360338956292
1398234331135500292
1400580861150613506
1400783012
14048767
14275715
1430446644571607045
1433791346
1439338194
```

```
#plot histogram of k-Core centrality
#plt a histogram of the data
plt.hist(list(k_core_centralities.values()))
#Add labels and a title to the plot
plt.xlabel('k_core_centrality for combined')
plt.ylabel('Yaxais')
plt.gca().get_yaxis().get_major_locator().set_params(integer=True)
plt.title('Histogram of combined k_corecentrality values for retweet')
#show the plot
plt.show()
```




```
# Input and output file paths for low-influence replies
input_file = '/content/drive/My Drive/Colab Notebooks/cs131/replies_nobots_uniq_lowinfluence.NONOUSER.txt'
sampled_file = '/content/drive/My Drive/Colab Notebooks/cs131/sampled_replies_lowinfluence.txt'

# Step 1: Create a random 1% sample of the adjacency list
sampling_rate = 0.01
with open(input_file, 'r') as infile, open(sampled_file, 'w') as outfile:
    for line in infile:
        if random.random() < sampling_rate:  # Keep 1% of lines randomly
            outfile.write(line)

print(f"Random sample saved to: {sampled_file}")

# Step 2: Load the sampled adjacency list into a NetworkX graph
G_low = nx.read_edgelist(sampled_file, delimiter='\t', create_using=nx.Graph())

# Step 3: Calculate K-Core Centrality (coreness)
k_core_low = nx.core_number(G_low)

# Step 4: Save K-Core Centrality to a file
k_core_df_low = pd.DataFrame(list(k_core_low.items()), columns=['Node', 'K-Core'])
k_core_df_low.to_csv('/content/drive/My Drive/Colab Notebooks/cs131/k_core_centrality_sampled_low_replies.tsv', sep='\t', index=False)

# Step 5: Plot histogram for K-Core Centrality
plt.figure(figsize=(10, 6))
plt.hist(k_core_low.values(), bins=20, color='lightgreen', edgecolor='black')
plt.xlabel('K-Core Centrality (Low-Influence Replies - Sampled Data)')
plt.ylabel('Frequency')
plt.title('Histogram of K-Core Centrality - Low-Influence Replies (Sampled)')
plt.gca().get_yaxis().get_major_locator().set_params(integer=True)  # Ensure y-axis has integer ticks
plt.show()
```

```
Random sample saved to: /content/drive/My Drive/Colab Notebooks/cs131/sampled_replies_lowinfluence.txt
```

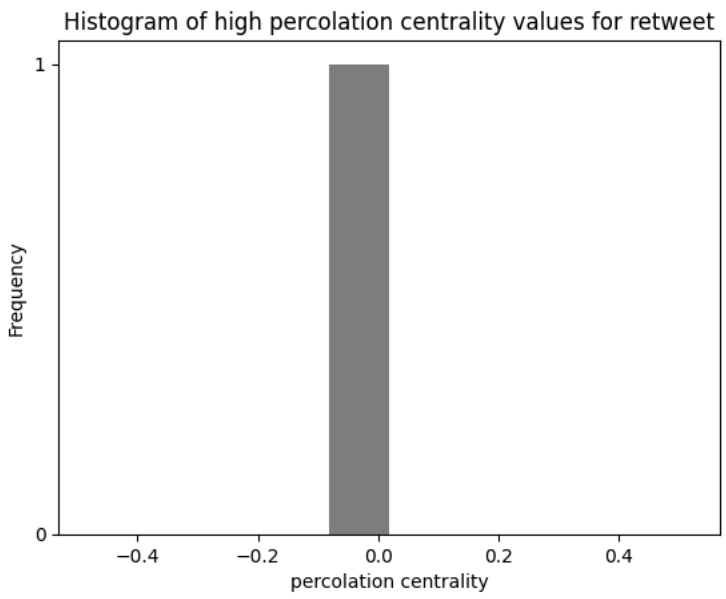


Percolation centrality

```
import networkx as nx

file_path = '/content/drive/My Drive/Colab Notebooks/cs131/retweets_nobots_uniq_highinfluence.NONOUSER.txt'

# Load the adjacency list
G = nx.read_edgelist(file_path, delimiter='\t')


#calculate the optimal percolation centralities
percolation_centralities={}
for node in G.nodes():
    #calculate the reduced degree centrality of all nodes at a distance  of 1 from the node
    reduced_degree_centrality=nx.degree_centrality(G)[node]/max(nx.degree_centrality(G).values())


#calculate the total reduced degree centrality of all nodees at a distance of 1 from the node
total_reduced_degree_centrality=1
for neighbor in G.neighbors(node):
    total_reduced_degree_centrality*=(1-nx.degree_centrality(G)[neighbor]/max(nx.degree_centrality(G).values()))


#calculate the percolation centralities
percolation_centrality=reduced_degree_centrality*total_reduced_degree_centrality


#Add the percolation centrality
percolation_centralities[node]=percolation_centrality


print(percolation_centralities[node])
```

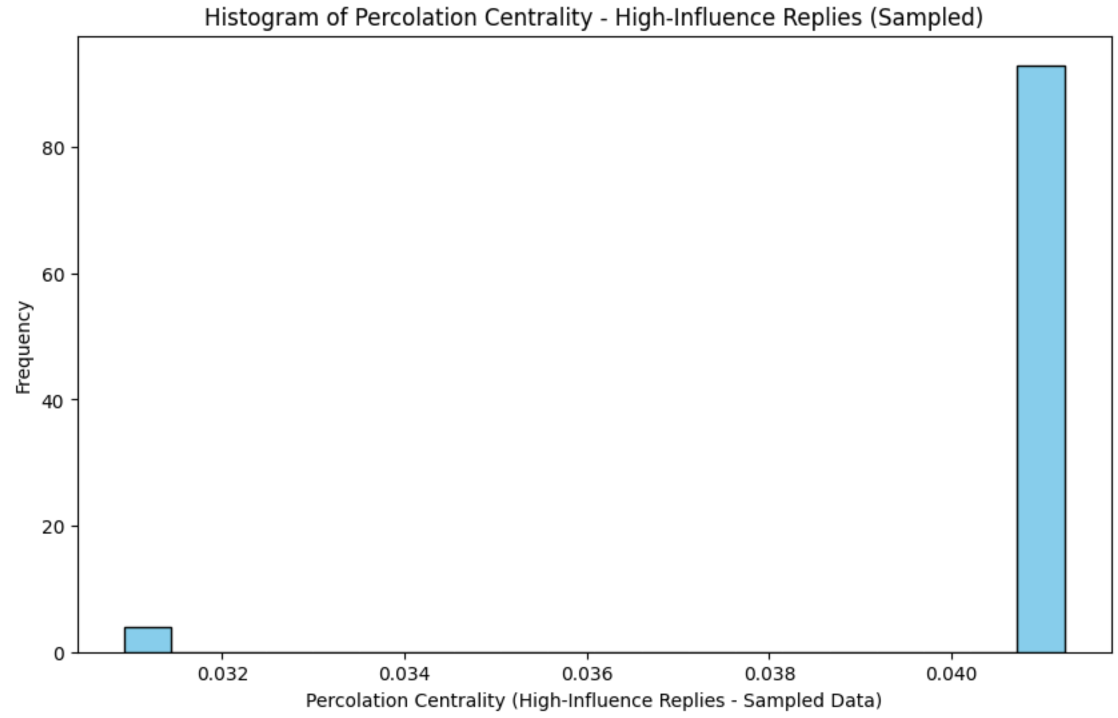
```
0.01872640328844575
```

```
#plt a histogram of the data
plt.hist(percolation_centralities.values(), bins=10,color='grey')
#Add labels and a title to the plot
plt.xlabel('percolation centrality')
plt.ylabel('Frequency')
plt.gca().get_yaxis().get_major_locator().set_params(integer=True)
plt.title('Histogram of high percolation centrality values for retweet')
#show the plot
plt.show()
```



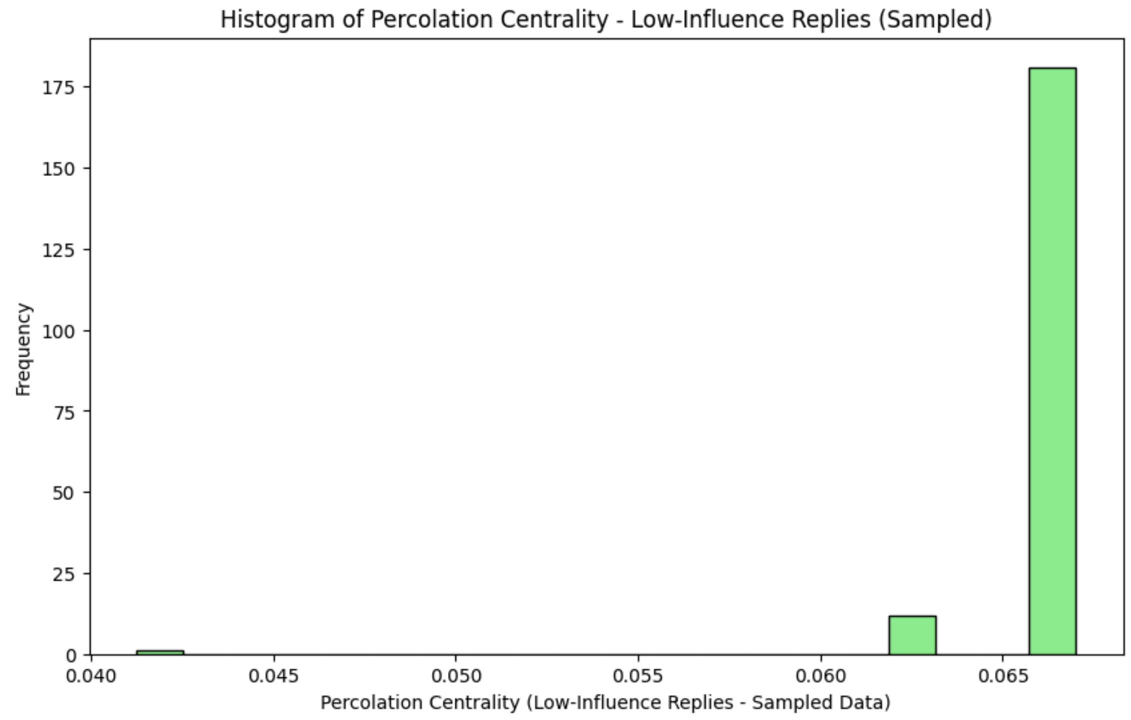

```
import random
import networkx as nx
import pandas as pd
import matplotlib.pyplot as plt

# Input and output file paths for high-influence replies
input_file = '/content/drive/My Drive/Colab Notebooks/cs131/replies_nobots_uniq_highinfluence.NONOUSER.txt'
sampled_file = '/content/drive/My Drive/Colab Notebooks/cs131/sampled_replies_highinfluence.txt'

# Step 1: Create a random 1% sample of the adjacency list
sampling_rate = 0.01
with open(input_file, 'r') as infile, open(sampled_file, 'w') as outfile:
    for line in infile:
        if random.random() < sampling_rate:  # Keep 1% of lines randomly
            outfile.write(line)

print(f"Random sample saved to: {sampled_file}")

# Step 2: Load the sampled adjacency list into a NetworkX graph
G_high = nx.read_edgelist(sampled_file, delimiter='\t', create_using=nx.Graph())

# Step 3: Calculate Percolation Centrality (approximated as node connectivity impact)
percolation_centrality_high = {}
for node in G_high:
    G_copy = G_high.copy()
    G_copy.remove_node(node)
    largest_component = max(nx.connected_components(G_copy), key=len, default=[])
    percolation_centrality_high[node] = len(largest_component) / len(G_high)

# Step 4: Save Percolation Centrality to a file
percolation_df_high = pd.DataFrame(list(percolation_centrality_high.items()), columns=['Node', 'Percolation Centrality'])
percolation_df_high.to_csv('/content/drive/My Drive/Colab Notebooks/cs131/percolation_centrality_sampled_high_replies.tsv', sep='\t', index=False)

# Step 5: Plot histogram for Percolation Centrality
plt.figure(figsize=(10, 6))
plt.hist(percolation_centrality_high.values(), bins=20, color='skyblue', edgecolor='black')
plt.xlabel('Percolation Centrality (High-Influence Replies - Sampled Data)')
plt.ylabel('Frequency')
plt.title('Histogram of Percolation Centrality - High-Influence Replies (Sampled)')
plt.gca().get_yaxis().get_major_locator().set_params(integer=True)  # Ensure y-axis has integer ticks
plt.show()
```

```
Random sample saved to: /content/drive/My Drive/Colab Notebooks/cs131/sampled_replies_highinfluence.txt
```




```
# Input and output file paths for low-influence replies
input_file = '/content/drive/My Drive/Colab Notebooks/cs131/replies_nobots_uniq_lowinfluence.NONOUSER.txt'
sampled_file = '/content/drive/My Drive/Colab Notebooks/cs131/sampled_replies_lowinfluence.txt'

# Step 1: Create a random 1% sample of the adjacency list
sampling_rate = 0.01
with open(input_file, 'r') as infile, open(sampled_file, 'w') as outfile:
    for line in infile:
        if random.random() < sampling_rate:  # Keep 1% of lines randomly
            outfile.write(line)

print(f"Random sample saved to: {sampled_file}")

# Step 2: Load the sampled adjacency list into a NetworkX graph
G_low = nx.read_edgelist(sampled_file, delimiter='\t', create_using=nx.Graph())

# Step 3: Calculate Percolation Centrality (approximated as node connectivity impact)
percolation_centrality_low = {}
for node in G_low:
    G_copy = G_low.copy()
    G_copy.remove_node(node)
    largest_component = max(nx.connected_components(G_copy), key=len, default=[])
    percolation_centrality_low[node] = len(largest_component) / len(G_low)

# Step 4: Save Percolation Centrality to a file
percolation_df_low = pd.DataFrame(list(percolation_centrality_low.items()), columns=['Node', 'Percolation Centrality'])
percolation_df_low.to_csv('/content/drive/My Drive/Colab Notebooks/cs131/percolation_centrality_sampled_low_replies.tsv', sep='\t', index=False)

# Step 5: Plot histogram for Percolation Centrality
plt.figure(figsize=(10, 6))
plt.hist(percolation_centrality_low.values(), bins=20, color='lightgreen', edgecolor='black')
plt.xlabel('Percolation Centrality (Low-Influence Replies - Sampled Data)')
plt.ylabel('Frequency')
plt.title('Histogram of Percolation Centrality - Low-Influence Replies (Sampled)')
plt.gca().get_yaxis().get_major_locator().set_params(integer=True)  # Ensure y-axis has integer ticks
plt.show()
```

```
Random sample saved to: /content/drive/My Drive/Colab Notebooks/cs131/sampled_replies_lowinfluence.txt
```

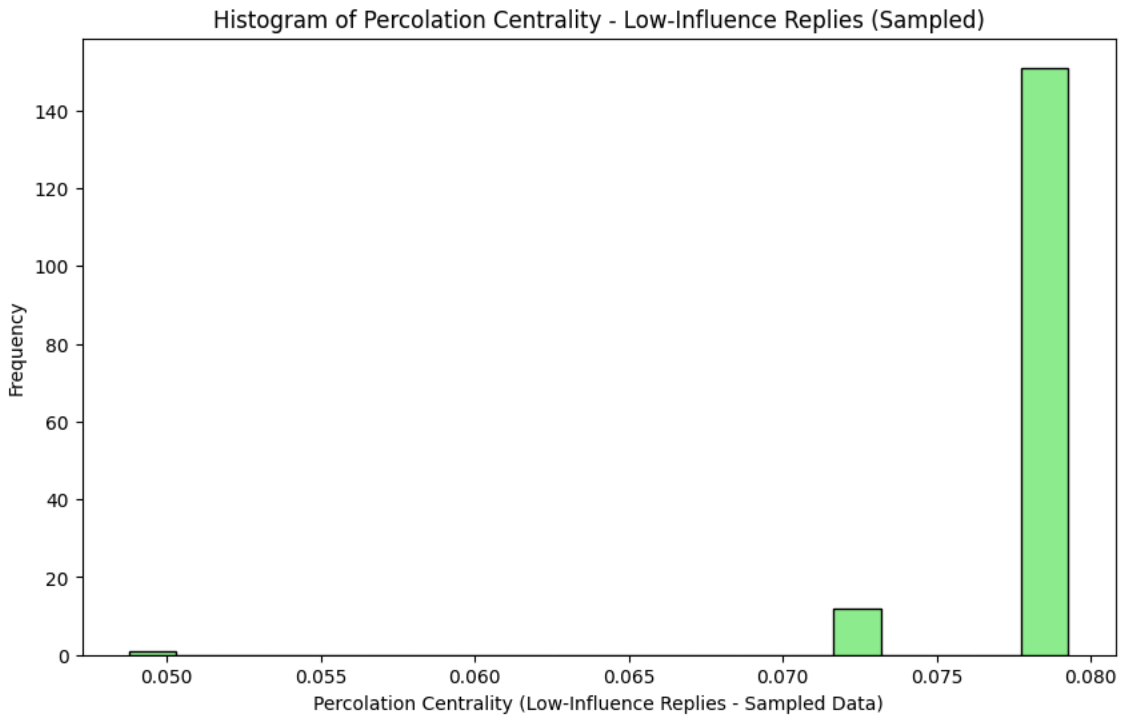



```
import networkx as nx

file_path = '/content/drive/My Drive/Colab Notebooks/cs131/retweets_nobots_uniq_lowinfluence.NONOUSER.txt'

# Load the adjacency list
G = nx.read_edgelist(file_path, delimiter='\t')
```

```
# Input and output file paths for low-influence replies
input_file = '/content/drive/My Drive/Colab Notebooks/cs131/replies_nobots_uniq_lowinfluence.NONOUSER.txt'
sampled_file = '/content/drive/My Drive/Colab Notebooks/cs131/sampled_replies_lowinfluence.txt'

# Step 1: Create a random 1% sample of the adjacency list
sampling_rate = 0.01
with open(input_file, 'r') as infile, open(sampled_file, 'w') as outfile:
    for line in infile:
        if random.random() < sampling_rate:  # Keep 1% of lines randomly
            outfile.write(line)
```

```
print(f"Random sample saved to: {sampled_file}")

# Step 2: Load the sampled adjacency list into a NetworkX graph
G_low = nx.read_edgelist(sampled_file, delimiter='\t', create_using=nx.Graph())

# Step 3: Calculate Percolation Centrality (approximated as node connectivity impact)
percolation_centrality_low = {}
for node in G_low:
    G_copy = G_low.copy()
    G_copy.remove_node(node)
    largest_component = max(nx.connected_components(G_copy), key=len, default=[])
    percolation_centrality_low[node] = len(largest_component) / len(G_low)

# Step 4: Save Percolation Centrality to a file
percolation_df_low = pd.DataFrame(list(percolation_centrality_low.items()), columns=['Node', 'Percolation Centrality'])
percolation_df_low.to_csv('/content/drive/My Drive/Colab Notebooks/cs131/percolation_centrality_sampled_low_replies.tsv', sep='\t', index=False)

# Step 5: Plot histogram for Percolation Centrality
plt.figure(figsize=(10, 6))
plt.hist(percolation_centrality_low.values(), bins=20, color='lightgreen', edgecolor='black')
plt.xlabel('Percolation Centrality (Low-Influence Replies - Sampled Data)')
plt.ylabel('Frequency')
plt.title('Histogram of Percolation Centrality - Low-Influence Replies (Sampled)')
plt.gca().get_yaxis().get_major_locator().set_params(integer=True)  # Ensure y-axis has integer ticks
plt.show()
```

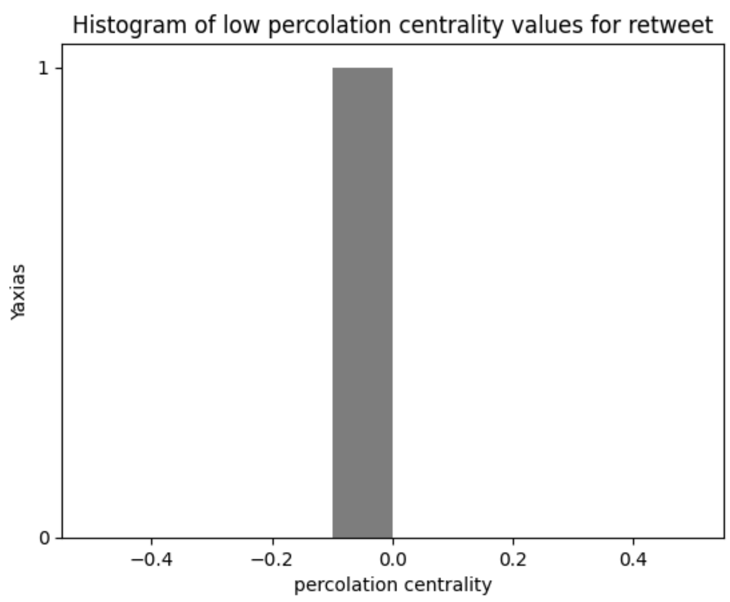
```
Random sample saved to: /content/drive/My Drive/Colab Notebooks/cs131/sampled_replies_lowinfluence.txt
```




```
#calculate the total reduced degree centrality of all nodees at a distance of 1 from the node
total_reduced_degree_centrality=1
for neighbor in G.neighbors(node):
    total_reduced_degree_centrality*=(1-nx.degree_centrality(G)[neighbor]/max(nx.degree_centrality(G).values()))


#calculate the percolation centralities
percolation_centrality=reduced_degree_centrality*total_reduced_degree_centrality


#Add the percolation centrality
percolation_centralities[node]=percolation_centrality


print(percolation_centralities[node])
```

```
0.001019608657074216
```

```
#plt a histogram of the data
plt.hist(percolation_centralities.values(), bins=10,color='grey')
#Add labels and a title to the plot
plt.xlabel('percolation centrality')
plt.ylabel('Yaxias')
plt.gca().get_yaxis().get_major_locator().set_params(integer=True)
plt.title('Histogram of low percolation centrality values for retweet')
#show the plot
plt.show()
```

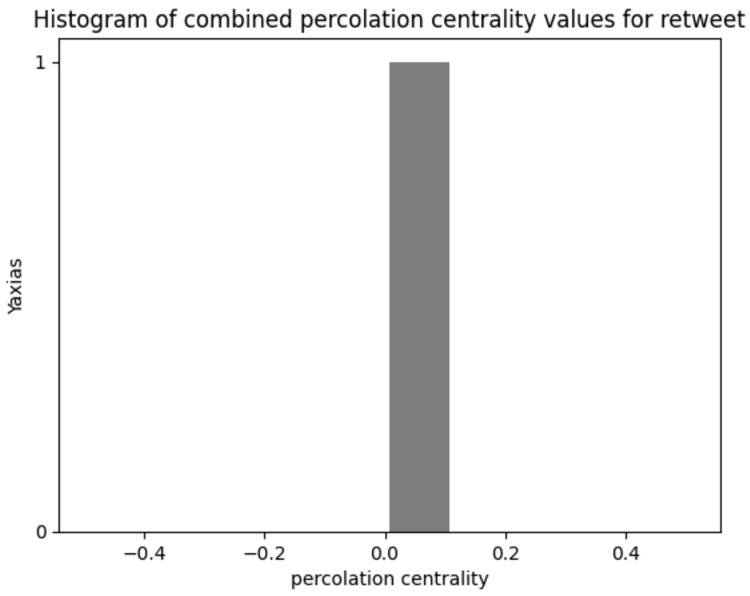



```
file_path = '/content/drive/My Drive/Colab Notebooks/cs131/combined_AdjList.tsv'

G = nx.read_edgelist(file_path, delimiter='\t')


# #calculate the optimal percolation centralities
# percolation_centralities={}
```

```
# for node in G.nodes():
#     #calculate the reduced degree centrality of all nodes at a distance  of 1 from the node
#     reduced_degree_centrality=nx.degree_centrality(G)[node]/max(nx.degree_centrality(G).values())


#calculate the total reduced degree centrality of all nodees at a distance of 1 from the node
total_reduced_degree_centrality=1
for neighbor in G.neighbors(node):
    total_reduced_degree_centrality*=(1-nx.degree_centrality(G)[neighbor]/max(nx.degree_centrality(G).values()))


#calculate the percolation centralities
percolation_centrality=reduced_degree_centrality*total_reduced_degree_centrality


#Add the percolation centrality
percolation_centralities[node]=percolation_centrality


print(percolation_centralities[node])
```

```
0.008040247300443136
```

```
#plt a histogram of the data
plt.hist(percolation_centralities.values(), bins=10,color='grey')
#Add labels and a title to the plot
plt.xlabel('percolation centrality')
plt.ylabel('Yaxias')
plt.gca().get_yaxis().get_major_locator().set_params(integer=True)
plt.title('Histogram of combined percolation centrality values for retweet')
#show the plot
plt.show()
```



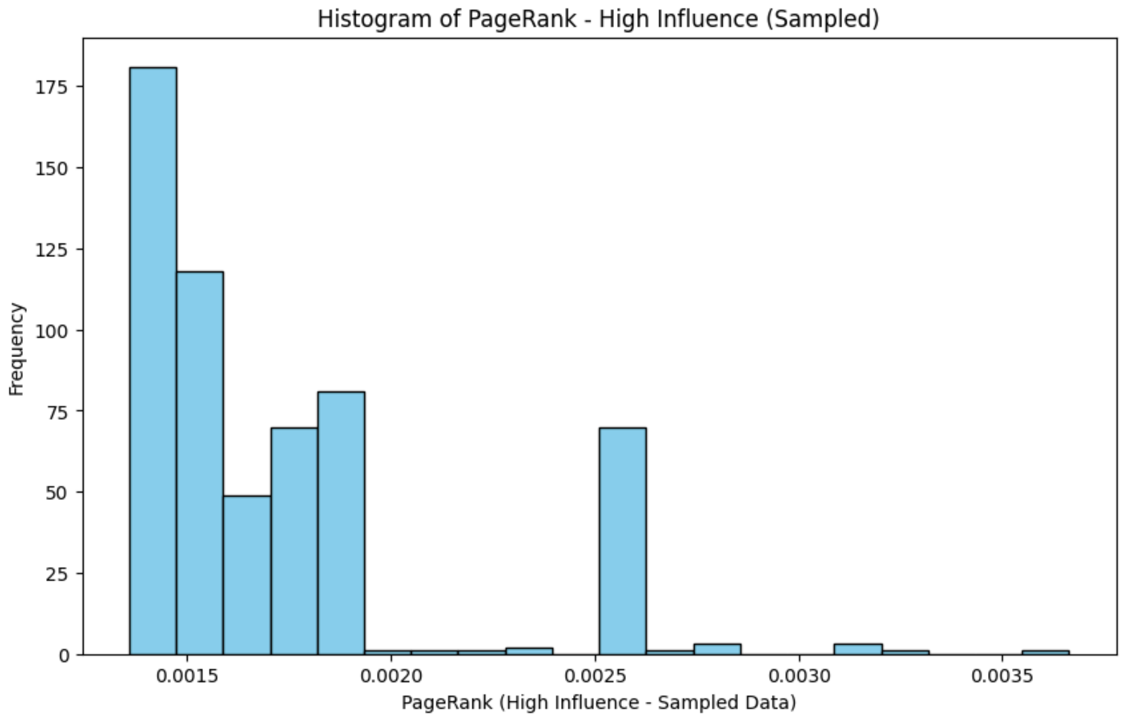
Spring Layouts

[ ] ↳ 4 cells hidden

PageRank

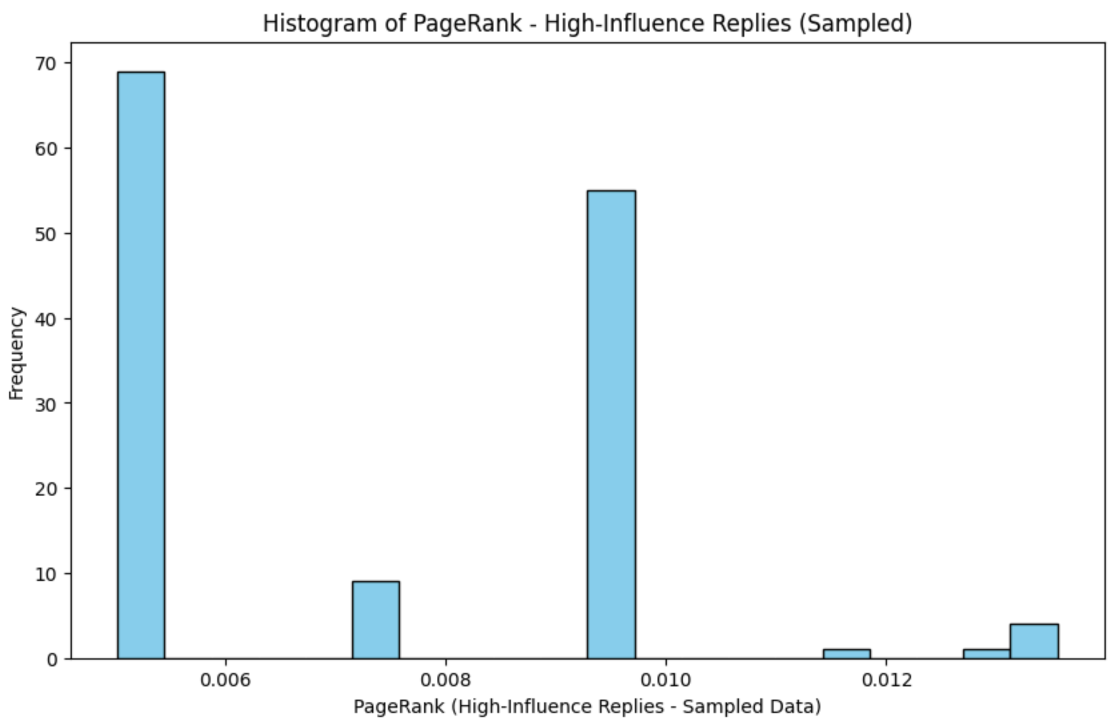
```
import random
import networkx as nx
import pandas as pd
import matplotlib.pyplot as plt

# Input and output file paths
input_file = '/content/drive/My Drive/Colab Notebooks/cs131/highinfl_AdjList.tsv'
sampled_file = '/content/drive/My Drive/Colab Notebooks/cs131/sampled_highinfl_AdjList.tsv'

# Step 1: Create a random 1% sample of the adjacency list
sampling_rate = 0.01
with open(input_file, 'r') as infile, open(sampled_file, 'w') as outfile:
    for line in infile:
        if random.random() < sampling_rate:  # Keep 1% of lines randomly
            outfile.write(line)

print(f"Random sample saved to: {sampled_file}")

# Step 2: Load the sampled adjacency list into a NetworkX graph
G = nx.read_edgelist(sampled_file, delimiter='\t', create_using=nx.DiGraph())

# Step 3: Calculate PageRank
pagerank = nx.pagerank(G)

# Step 4: Plot histogram for PageRank
plt.figure(figsize=(10, 6))
plt.hist(pagerank.values(), bins=20, color='skyblue', edgecolor='black')
plt.xlabel('PageRank (High Influence - Sampled Data)')
plt.ylabel('Frequency')
plt.gca().get_yaxis().get_major_locator().set_params(integer=True)  # Ensure y-axis has integer ticks
plt.title('Histogram of PageRank - High Influence (Sampled)')
plt.show()
```

Random sample saved to: /content/drive/My Drive/Colab Notebooks/cs131/sampled_highinfl_AdjList.tsv




```
import random
import networkx as nx
import pandas as pd
import matplotlib.pyplot as plt

# Input and output file paths for high-influence replies
input_file = '/content/drive/My Drive/Colab Notebooks/cs131/replies_nobots_uniq_highinfluence.NONOUSER.txt'
sampled_file = '/content/drive/My Drive/Colab Notebooks/cs131/sampled_replies_highinfluence.txt'

# Step 1: Create a random 1% sample of the adjacency list
sampling_rate = 0.01
with open(input_file, 'r') as infile, open(sampled_file, 'w') as outfile:
    for line in infile:
        if random.random() < sampling_rate:  # Keep 1% of lines randomly
            outfile.write(line)

print(f"Random sample saved to: {sampled_file}")

# Step 2: Load the sampled adjacency list into a NetworkX graph
G_high = nx.read_edgelist(sampled_file, delimiter='\t', create_using=nx.DiGraph())

# Step 3: Calculate PageRank
pagerank_high = nx.pagerank(G_high)

# Step 4: Save PageRank to a file
pagerank_df_high = pd.DataFrame(list(pagerank_high.items()), columns=['Node', 'PageRank'])
pagerank_df_high.to_csv('/content/drive/My Drive/Colab Notebooks/cs131/pagerank_sampled_high_replies.tsv', sep='\t', index=False)

# Step 5: Plot histogram for PageRank
plt.figure(figsize=(10, 6))
plt.hist(pagerank_high.values(), bins=20, color='skyblue', edgecolor='black')
plt.xlabel('PageRank (High-Influence Replies - Sampled Data)')
plt.ylabel('Frequency')
plt.title('Histogram of PageRank - High-Influence Replies (Sampled)')
plt.gca().get_yaxis().get_major_locator().set_params(integer=True)  # Ensure y-axis has integer ticks
plt.show()
```

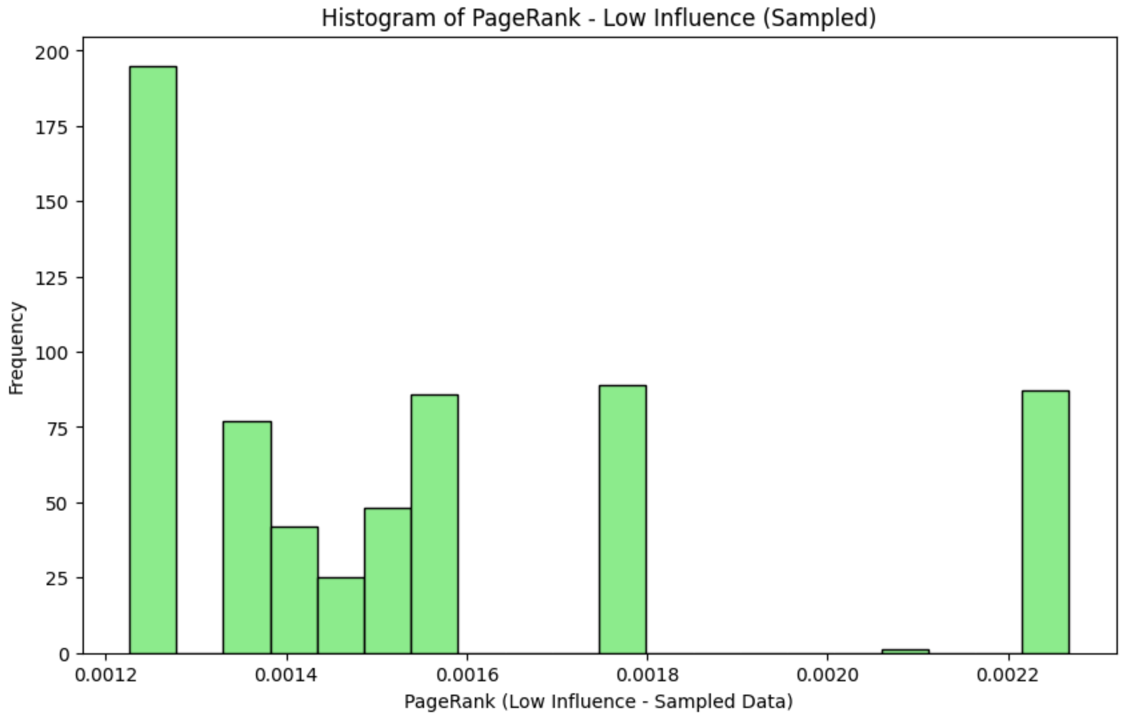
Random sample saved to: /content/drive/My Drive/Colab Notebooks/cs131/sampled_replies_highinfluence.txt




```
import random
import networkx as nx
import pandas as pd
import matplotlib.pyplot as plt

# Input and output file paths
input_file = '/content/drive/My Drive/Colab Notebooks/cs131/lowinfl_AdjList.tsv'
sampled_file = '/content/drive/My Drive/Colab Notebooks/cs131/sampled_lowinfl_AdjList.tsv'

# Step 1: Create a random 1% sample of the adjacency list
sampling_rate = 0.01
with open(input_file, 'r') as infile, open(sampled_file, 'w') as outfile:
    for line in infile:
        if random.random() < sampling_rate:  # Keep 1% of lines randomly
            outfile.write(line)

print(f"Random sample saved to: {sampled_file}")
```

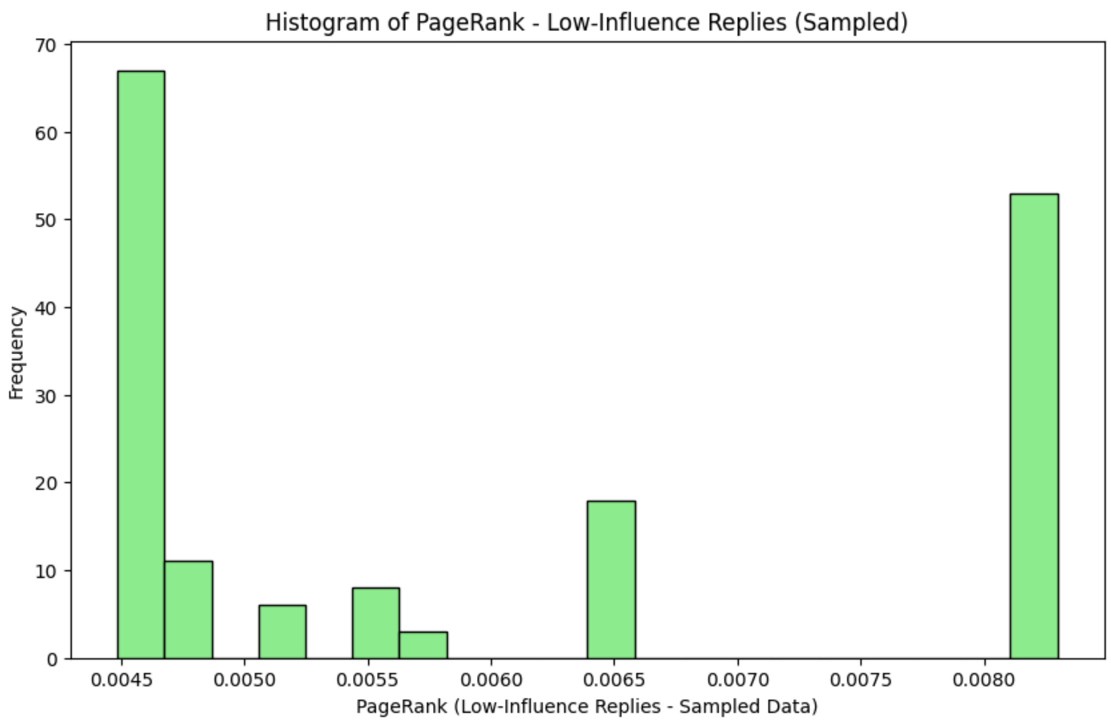
```
# Step 2: Load the sampled adjacency list into a NetworkX graph
G = nx.read_edgelist(sampled_file, delimiter='\t', create_using=nx.DiGraph())

# Step 3: Calculate PageRank
pagerank = nx.pagerank(G)

# Step 4: Plot histogram for PageRank
plt.figure(figsize=(10, 6))
plt.hist(pagerank.values(), bins=20, color='lightgreen', edgecolor='black')
plt.xlabel('PageRank (Low Influence - Sampled Data)')
plt.ylabel('Frequency')
plt.gca().get_yaxis().get_major_locator().set_params(integer=True)  # Ensure y-axis has integer ticks
plt.title('Histogram of PageRank - Low Influence (Sampled)')
plt.show()
```

```
Random sample saved to: /content/drive/My Drive/Colab Notebooks/cs131/sampled_lowinfl_AdjList.tsv
```




```
# Input and output file paths for low-influence replies
input_file = '/content/drive/My Drive/Colab Notebooks/cs131/replies_nobots_uniq_lowinfluence.NONOUSER.txt'
sampled_file = '/content/drive/My Drive/Colab Notebooks/cs131/sampled_replies_lowinfluence.txt'

# Step 1: Create a random 1% sample of the adjacency list
sampling_rate = 0.01
with open(input_file, 'r') as infile, open(sampled_file, 'w') as outfile:
    for line in infile:
        if random.random() < sampling_rate:  # Keep 1% of lines randomly
            outfile.write(line)

print(f"Random sample saved to: {sampled_file}")

# Step 2: Load the sampled adjacency list into a NetworkX graph
G_low = nx.read_edgelist(sampled_file, delimiter='\t', create_using=nx.DiGraph())

# Step 3: Calculate PageRank
pagerank_low = nx.pagerank(G_low)

# Step 4: Save PageRank to a file
pagerank_df_low = pd.DataFrame(list(pagerank_low.items()), columns=['Node', 'PageRank'])
pagerank_df_low.to_csv('/content/drive/My Drive/Colab Notebooks/cs131/pagerank_sampled_low_replies.tsv', sep='\t', index=False)

# Step 5: Plot histogram for PageRank
plt.figure(figsize=(10, 6))
plt.hist(pagerank_low.values(), bins=20, color='lightgreen', edgecolor='black')
plt.xlabel('PageRank (Low-Influence Replies - Sampled Data)')
plt.ylabel('Frequency')
plt.title('Histogram of PageRank - Low-Influence Replies (Sampled)')
plt.gca().get_yaxis().get_major_locator().set_params(integer=True)  # Ensure y-axis has integer ticks
plt.show()
```

```
Random sample saved to: /content/drive/My Drive/Colab Notebooks/cs131/sampled_replies_lowinfluence.txt
```



Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.