LAB MANUAL
PART A
(PART A: TO BE REFFERED BY STUDENTS)

Experiment No-06

A.1 Aim:

Develop Structure based social media analytics model for any business. (e.g. Structure Based Models - community detection, influence analysis)

Lab Objective	To design and develop social media analytics models
Lab Outcome	Design and develop content and structure based social media analytics models

A-2 Prerequisite

Python

A.3 OutCome

Students will be able

A.4 Theory:

Structure Based Analysis:

The rise of social media has turned the Web into an online community where people connect, communicate, and collaborate with each other. Structured analytics in social media is the process of discovering the structure of the relationships emerging from this social media use. It focuses on identifying the users involved, the activities they undertake, the actions they perform, and the items (e.g., movies, restaurants, blogs, etc.) they create and interact with. There are two key challenges facing these tasks: how to organize and model social media content, which is often unstructured in its raw form, in order to employ structured analytics on it; and how to employ analytics algorithms to capture both explicit link-based relationships and implicit behavior-based relationships. In this tutorial, we systemize and summarize the research so far in analyzing social interactions between users and items in the Web from data mining and database perspectives. We start with a general overview of the topic, including discourse to various exciting and practical applications. Then, we discuss the state-of-art for modeling the data, formalizing the mining task, developing the algorithmic solutions, and evaluating on real datasets. We also emphasize open problems and challenges for future research in the area of structured analytics and social media.

Community Detection

Community detection is a fundamental problem in social network analysis consisting, roughly speaking, in unsupervised dividing social actors (modeled as nodes in a social graph) with certain social connections (modeled as edges in the social graph) into densely knitted and highly related groups with each group well separated from the others. Classical approaches for community detection usually deal only with the structure of the network and ignore features of the nodes (traditionally called node attributes), although the majority of real-world social networks provide additional actors' information such as age, gender, interests, etc. It is believed that the attributes may clarify and enrich the knowledge about the actors and give sense to the detected communities. This belief has motivated the progress in developing community detection methods that use both the structure and the attributes of the network (modeled already via a node-attributed graph) to yield more informative and qualitative community detection results.

Why Community Detection?

When analyzing different networks, it may be important to discover communities inside them. Community detection techniques are useful for social media algorithms to discover people with common interests and keep them tightly connected. Community detection can be used in machine learning to detect groups with similar properties and extract groups for various reasons. For example, this technique can be used to discover manipulative groups inside a social network or a stock market. Community detection is very applicable in understanding and evaluating the structure of large and complex networks. This approach uses the properties of edges in graphs or networks and hence more suitable for network analysis rather than a clustering approach. The clustering algorithms have a tendency to separate single peripheral nodes from the communities it should belong to. Many different algorithms have proposed and implemented for network community detection. Each of these has various pros and cons depending on the nature of the network as well as the applying problem domain.

Influence Analysis

A set of techniques that allows one to determine the degree to which specific data points affect the overall result of a statistical procedure. Social influence analysis (SIA) is becoming an important research field in social networks. SIA mainly studies how to model the influence diffusion process in networks, and how to propose an efficient method to identify a group of target nodes in a network.

Network Analysis

Network analysis (NA) is a set of integrated techniques to depict relations among actors and to analyze the social structures that emerge from the recurrence of these relations. The basic assumption is that better explanations of social phenomena are yielded by analysis of the relations among entities.

PART B

(PART B: TO BE COMPLETED BY STUDENTS)

Roll. No. C36	**Name: Sanskruti Kadam**
Class: BE	**Batch: C2**
Date of Experiment:	**Date of Submission:**
Grade:	

B.1. Study the fundamentals of community detection and influence analysis and develop structure based social media analytics model for any business:
(Paste your Search material completed during the 2 hours of practical in the lab here)

1. Community Detection
2. Influence Analysis
3. (NOTE: anyone can be the part of this experiment out of 2)

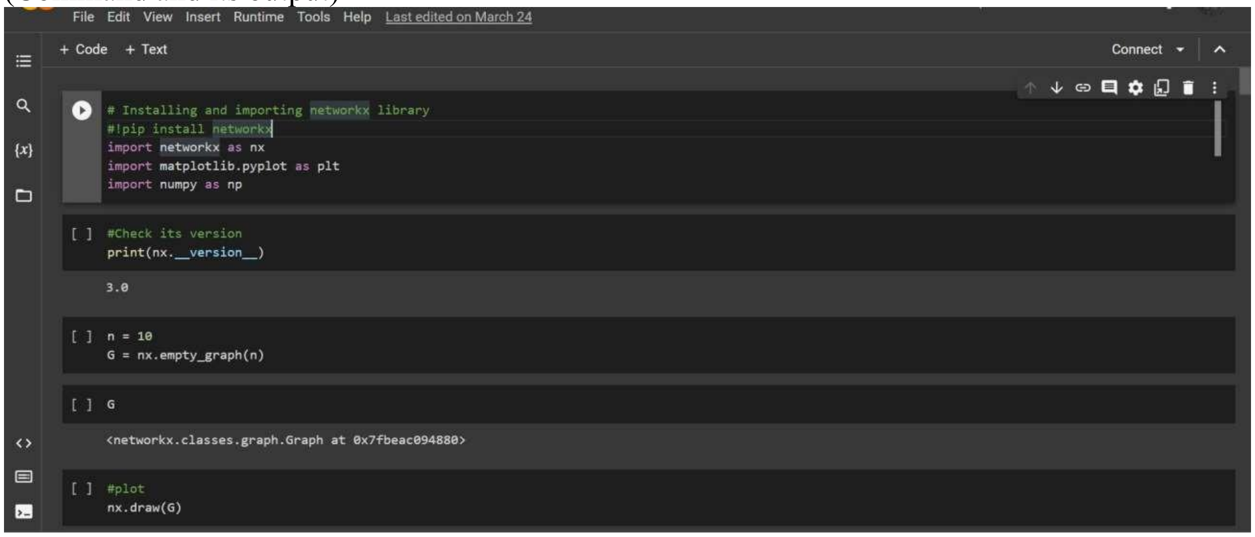
```
# Installing and importing networkx library #!pip install networkx
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np

#Check its version
print(nx. version )

n = 10
G = nx.empty_graph(n)
G
#plot
nx.draw(G)

# Lets add vetices labels
nx.draw(G, with_labels=True)

K_5 = nx.complete_graph(5)
nx.draw(K_5, with_labels=True)
nx.draw(K_5, with_labels=True)
K_5_D = nx.DiGraph(K_5)
nx.draw(K_5_D, with_labels=True)
st = nx.star_graph(20)


nx.draw(st, with_labels=True)

rt = nx.random_tree(20)
nx.draw(rt, with_labels=True)

bt = nx.balanced_tree(3,2)
nx.draw(bt, with_labels=True)
```

```
er = nx.erdos_renyi_graph(100, 0.15) #100 nodes connected with probability 0.15
nx.draw(er, with_labels=True)

#Degree plot
degrees = [er.degree(n) for n in er.nodes()]
plt.hist(degrees)

%matplotlib inline
import pandas as pd
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
from random import randint

facebook = pd.read_csv( "/content/facebook_combined.txt.gz", compression="gzip",
sep=" ",names=["start_node", "end_node"],)
facebook

G = nx.from_pandas_edgelist(facebook, "start_node", "end_node")


fig, ax = plt.subplots(figsize=(15, 9))
ax.axis("off")
plot_options = {"node_size": 10, "with_labels": False, "width": 0.15}
nx.draw_networkx(G, pos=nx.random_layout(G), ax=ax, **plot_options)

pos = nx.spring_layout(G, iterations=15, seed=1721)
fig, ax = plt.subplots(figsize=(15, 9))
ax.axis("off")
nx.draw_networkx(G, pos=pos, ax=ax, **plot_options)
G.number_of_nodes()

G.number_of_edges()

np.mean([d for _, d in G.degree()])

shortest_path_lengths = dict(nx.all_pairs_shortest_path_length(G))
shortest_path_lengths[0][42] # Length of shortest path between nodes 0 and 42
# This is equivalent to `diameter = nx.diameter(G), but much more efficient sinc
e we're # reusing the pre-computed shortest path lengths!
diameter = max(nx.eccentricity(G, sp=shortest_path_lengths).values())
diameter

# Compute the average shortest path length for each node
average_path_lengths = [np.mean(list(spl.values())) for spl in shortest_path_len
gths.values()]
# The average over all nodes
np.mean(average_path_lengths)
```

```
# We know the maximum shortest path length (the diameter), so create an array #
to store values from 0 up to (and including) diameter
path_lengths = np.zeros(diameter + 1, dtype=int)

# Extract the frequency of shortest path lengths between two nodes for pls in sh
ortest_path_lengths.values():
pl, cnts = np.unique(list(pls.values()), return_counts=True)
path_lengths[pl] += cnts

# Express frequency distribution as a percentage (ignoring path lengths of 0) fr
eq_percent = 100 * path_lengths[1:] / path_lengths[1:].sum()

# Plot the frequency distribution (ignoring path lengths of 0) as a percentage f
ig, ax = plt.subplots(figsize=(15, 8))
ax.bar(np.arange(1, diameter + 1), height=freq_percent)
ax.set_title("Distribution of shortest path length in G", fontdict={"size": 35},
 loc="center")
ax.set_xlabel("Shortest Path Length", fontdict={"size": 22})
ax.set_ylabel("Frequency (%)", fontdict={"size": 22})

nx.density(G)
nx.number_connected_components(G)
degree_centrality = nx.centrality.degree_centrality(G)
# save results in a variable to use again
(sorted(degree_centrality.items(), key=lambda item: item[1], reverse=True))[:8]
(sorted(G.degree, key=lambda item: item[1], reverse=True))[:8]
plt.figure(figsize=(15, 8))
plt.hist(degree_centrality.values(), bins=25)
plt.xticks(ticks=[0, 0.025, 0.05, 0.1, 0.15, 0.2]) # set the x axis ticks
plt.title("Degree Centrality Histogram ", fontdict={"size": 35}, loc="center")
plt.xlabel("Degree Centrality", fontdict={"size": 20})
plt.ylabel("Counts", fontdict={"size": 20})

node_size = [v * 1000 for v in degree_centrality.values()]
# set up nodes size for a nice graph representation
plt.figure(figsize=(15, 8))
nx.draw_networkx(G, pos=pos, node_size=node_size, with_labels=False, width=0.15)

plt.axis("off")

betweenness_centrality = nx.centrality.betweenness_centrality(G) # save results
in a variable to use again
(sorted(betweenness_centrality.items(), key=lambda item: item[1], reverse=True))
[:8]

plt.figure(figsize=(15, 8))
plt.hist(betweenness_centrality.values(), bins=100)
plt.xticks(ticks=[0, 0.02, 0.1, 0.2, 0.3, 0.4, 0.5])
# set the x axis ticks
```

```
plt.title("Betweenness Centrality Histogram ", fontdict={"size": 35}, loc="cente
r")
plt.xlabel("Betweenness Centrality", fontdict={"size": 20})
plt.ylabel("Counts", fontdict={"size": 20})

node_size = [v * 1200 for v in betweenness_centrality.values()]
# set up nodes size for a nice graph representation
plt.figure(figsize=(15, 8))
nx.draw_networkx(G, pos=pos, node_size=node_size, with_labels=False, width=0.15)

plt.axis("off")

closeness_centrality = nx.centrality.closeness_centrality(G) # save results in a
 variable to use again
(sorted(closeness_centrality.items(), key=lambda item: item[1], reverse=True))[:
8]
1 / closeness_centrality[107]

plt.figure(figsize=(15, 8))
plt.hist(closeness_centrality.values(), bins=60)
plt.title("Closeness Centrality Histogram ", fontdict={"size": 35}, loc="center"
)
plt.xlabel("Closeness Centrality", fontdict={"size": 20})
plt.ylabel("Counts", fontdict={"size": 20})

node_size = [v * 50 for v in closeness_centrality.values()]
# set up nodes size for a nice graph representation
plt.figure(figsize=(15, 8))
nx.draw_networkx(G, pos=pos, node_size=node_size, with_labels=False, width=0.15)

plt.axis("off")

eigenvector_centrality = nx.centrality.eigenvector_centrality(G)
# save results in a variable to use again
(sorted(eigenvector_centrality.items(), key=lambda item: item[1], reverse=True))
[:10]

high_eigenvector_centralities = (sorted(eigenvector_centrality.items(), key=lamb
da item: item[1], reverse=True))[1:10]
# 2nd to 10th nodes with heighest eigenvector centralities
high_eigenvector_nodes = [tuple[0] for tuple in high_eigenvector_centralities]
# set list as [2266, 2206, 2233, 2464, 2142, 2218, 2078, 2123, 1993]
neighbors_1912 = [n for n in G.neighbors(1912)] # list with all nodes connected
to 1912
all(item in neighbors_1912 for item in high_eigenvector_nodes)
# check if items in list high_eigenvector_nodes exist in list neighbors_1912

plt.figure(figsize=(15, 8))
plt.hist(eigenvector_centrality.values(), bins=60)
plt.xticks(ticks=[0, 0.01, 0.02, 0.04, 0.06, 0.08]) # set the x axis ticks plt.t
itle("Eigenvector Centrality Histogram ", fontdict={"size": 35}, loc="center") p
lt.xlabel("Eigenvector Centrality", fontdict={"size": 20})
plt.ylabel("Counts", fontdict={"size": 20})

node_size = [
v * 4000 for v in eigenvector_centrality.values()
] # set up nodes size for a nice graph representation plt.figure(figsize=(15, 8)
)
nx.draw_networkx(G, pos=pos, node_size=node_size, with_labels=False, width=0.15)
 plt.axis("off")

nx.average_clustering(G)


plt.figure(figsize=(15, 8)) plt.hist(nx.clustering(G).values(), bins=50)
plt.title("Clustering Coefficient Histogram ", fontdict={"size": 35}, loc="cente
r") plt.xlabel("Clustering Coefficient", fontdict={"size": 20})
plt.ylabel("Counts", fontdict={"size": 20})

triangles_per_node = list(nx.triangles(G).values())
sum(triangles_per_node
) / 3 # divide by 3 because each triangle is counted once for each node
np.mean(triangles_per_node)
np.median(triangles_per_node) nx.has_bridges(G)
bridges = list(nx.bridges(G)) len(bridges)

local_bridges = list(nx.local_bridges(G, with_span=False)) len(local_bridges)

plt.figure(figsize=(15, 8))
nx.draw_networkx(G, pos=pos, node_size=10, with_labels=False, width=0.15) nx.dra
w_networkx_edges(
G, pos, edgelist=local_bridges, width=0.5, edge_color="lawngreen"
) # green color for local bridges nx.draw_networkx_edges(G,
pos, edgelist=bridges, width=0.5, edge_color="r"
) # red color for bridges plt.axis("off")
nx.degree_assortativity_coefficient(G) nx.degree_pearson_correlation_coefficient
(
G
) # use the potentially faster scipy.stats.pearsonr function.

colors = ["" for x in range(G.number_of_nodes())] # initialize colors list count
er = 0
for com in nx.community.label_propagation_communities(G):
color = "#%06X" % randint(0, 0xFFFFFF) # creates random RGB color counter += 1
for node in list(

com
): # fill colors list with the particular color for the community nodes colors[n
ode] = color
counter

plt.figure(figsize=(15, 9)) plt.axis("off") nx.draw_networkx(
G, pos=pos, node_size=10, with_labels=False, width=0.15, node_color=colors)

colors = ["" for x in range(G.number_of_nodes())] for com in nx.community.asyn_f
luidc(G, 8, seed=0):
color = "#%06X" % randint(0, 0xFFFFFF) # creates random RGB color
for node in list(com):
  colors[node] = color

plt.figure(figsize=(15, 9)) plt.axis("off") nx.draw_networkx(G, pos=pos, node_si
ze=10, with_labels=False, width=0.15, node_color=colors)
```

B.2 Input and Output:
(Command and its output)


File Edit View Insert Runtime Tools Help Last edited on March 24
+ Code + Text
Connect

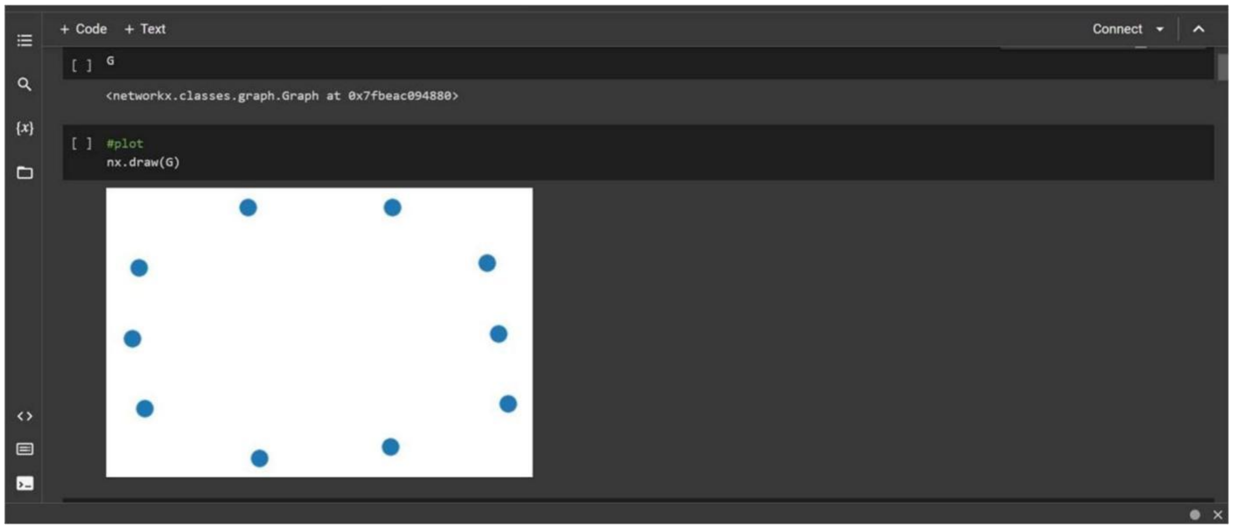
```
# Installing and importing networkx library
#!pip install networkx
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np
```

```
[ ] #Check its version
    print(nx.__version__)

    3.0
```

```
[ ] n = 10
    G = nx.empty_graph(n)
```

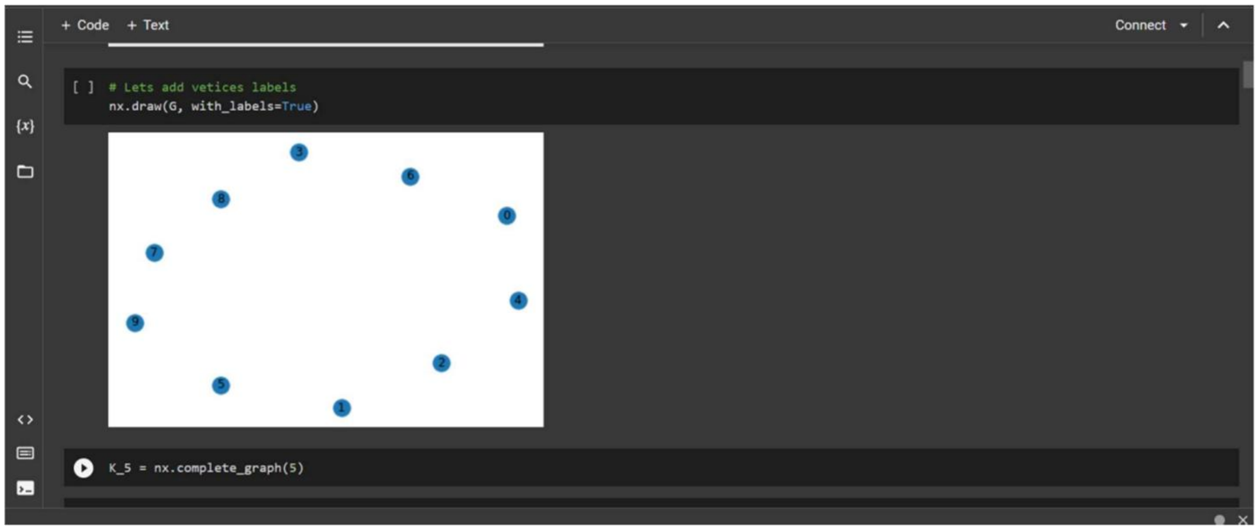
```
[ ] G

    <networkx.classes.graph.Graph at 0x7fbeac094880>
```

```
[ ] #plot
    nx.draw(G)
```



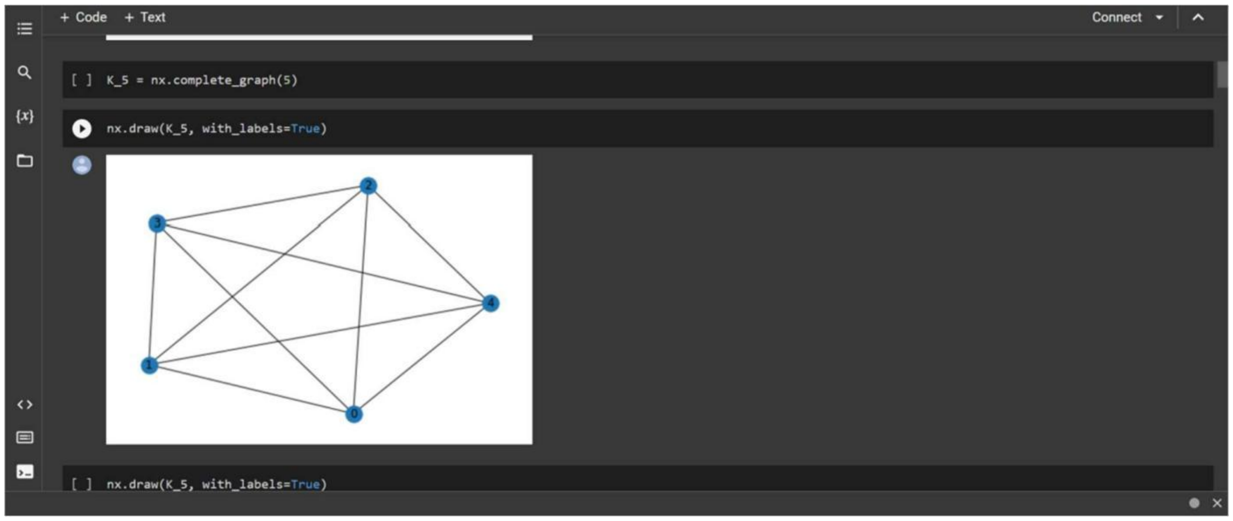
```
[ ] G

    <networkx.classes.graph.Graph at 0x7fbeac094880>
```

```
[ ] #plot
    nx.draw(G)
```



```
[ ] # Lets add vetices labels
    nx.draw(G, with_labels=True)
```

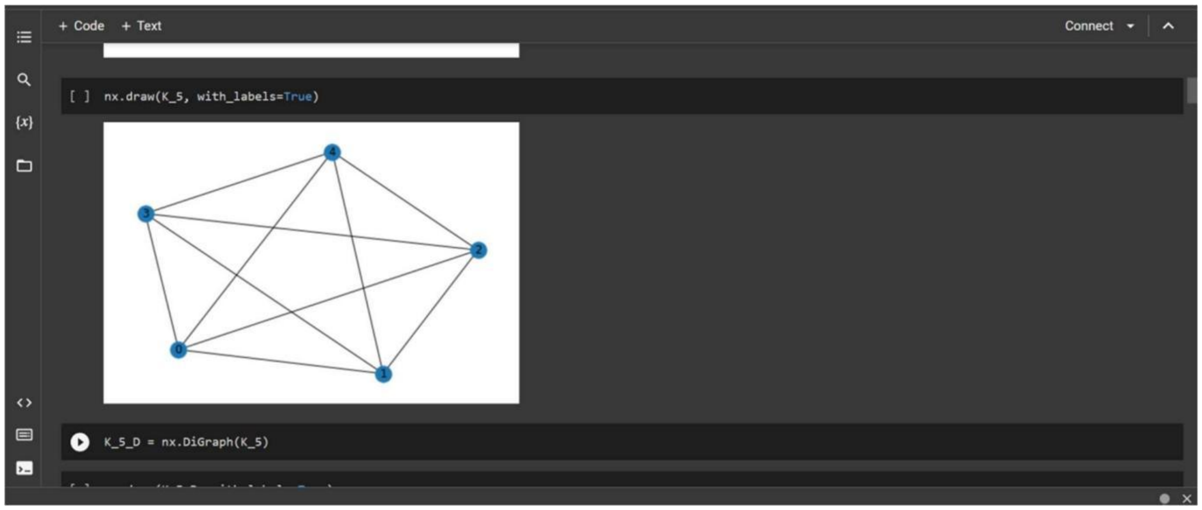
```
K_5 = nx.complete_graph(5)
```



```
[ ] K_5 = nx.complete_graph(5)
```

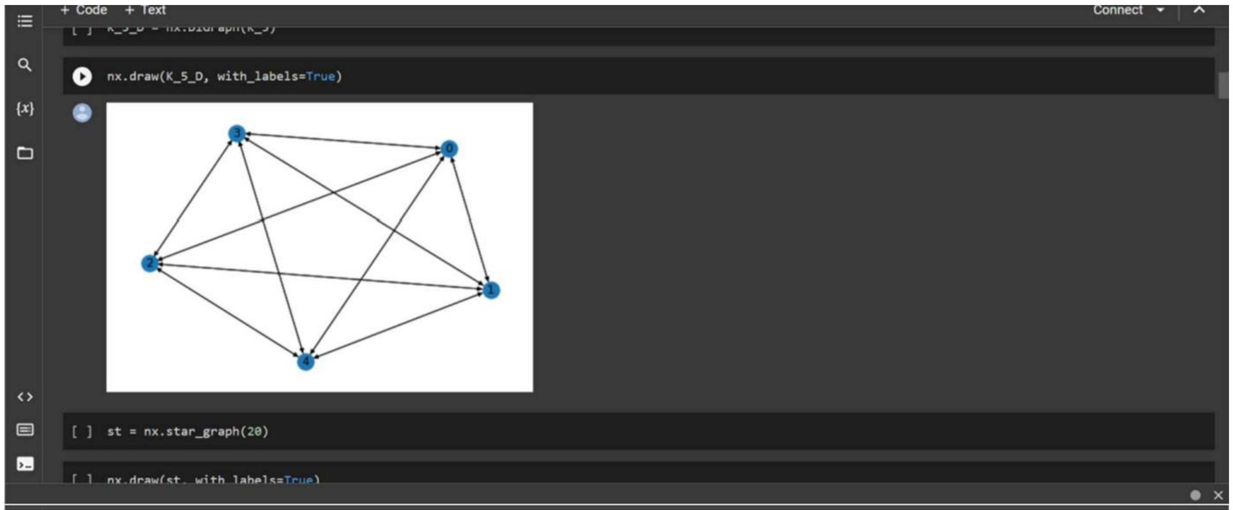
```
nx.draw(K_5, with_labels=True)
```

```
[ ] nx.draw(K_5, with_labels=True)
```



```
[ ] nx.draw(K_5, with_labels=True)
```

```
K_5_D = nx.DiGraph(K_5)
```

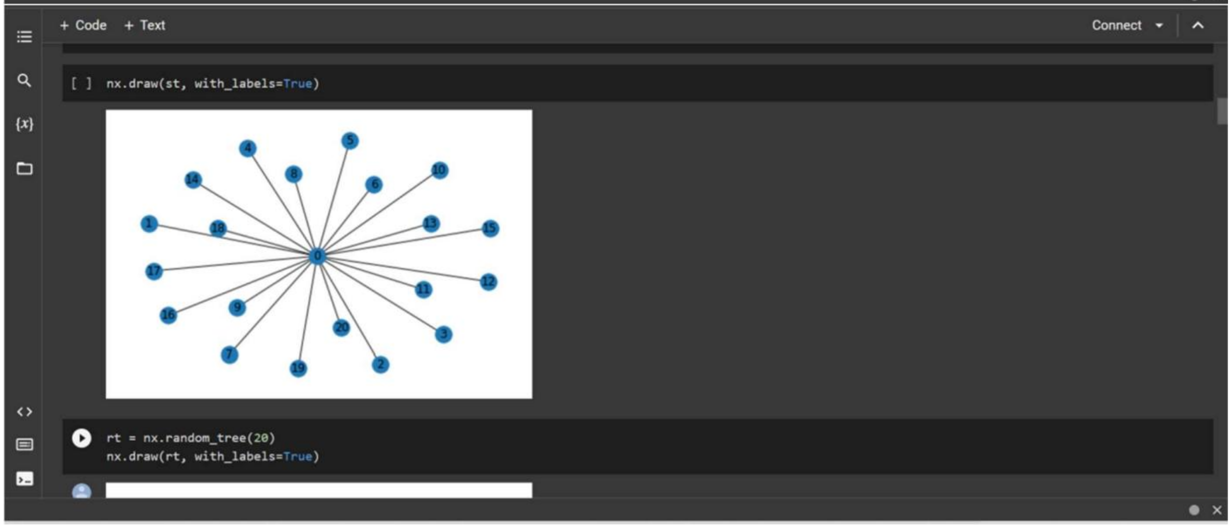


```
[ ] K_5_D = nx.DiGraph(K_5)
```

```
nx.draw(K_5_D, with_labels=True)
```

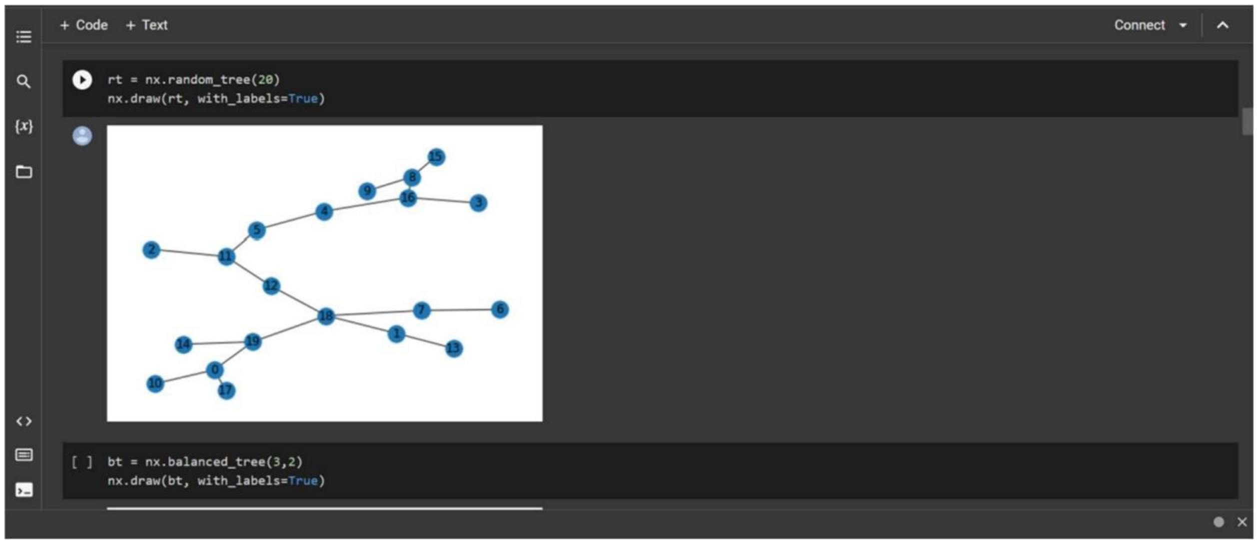
```
[ ] st = nx.star_graph(20)
```

```
[ ] nx.draw(st, with_labels=True)
```



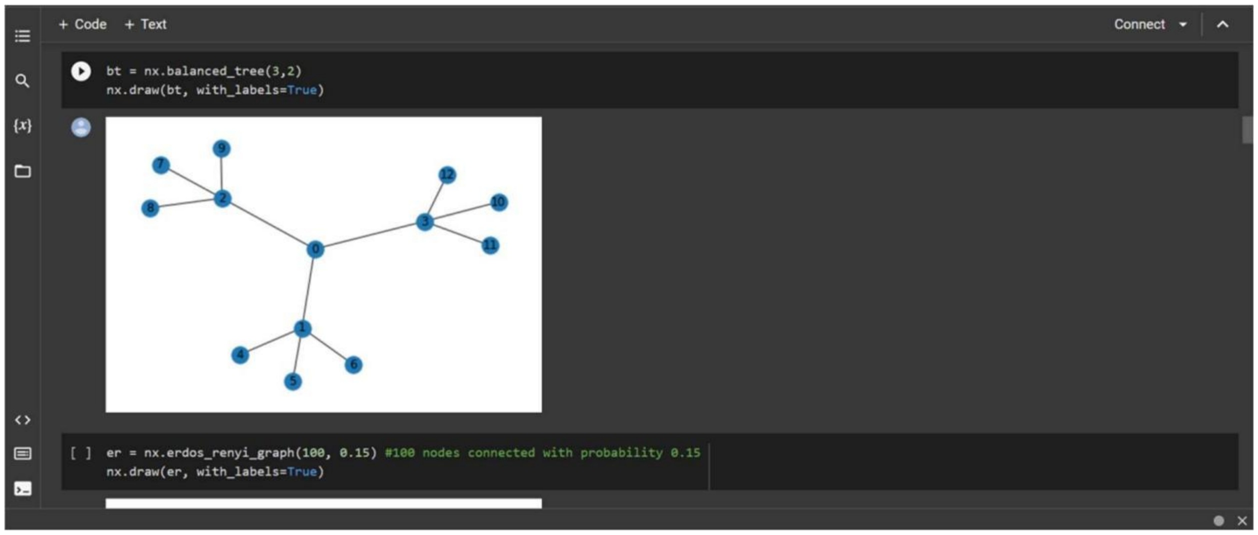
```
[ ] nx.draw(st, with_labels=True)
```

```
rt = nx.random_tree(20)
nx.draw(rt, with_labels=True)
```



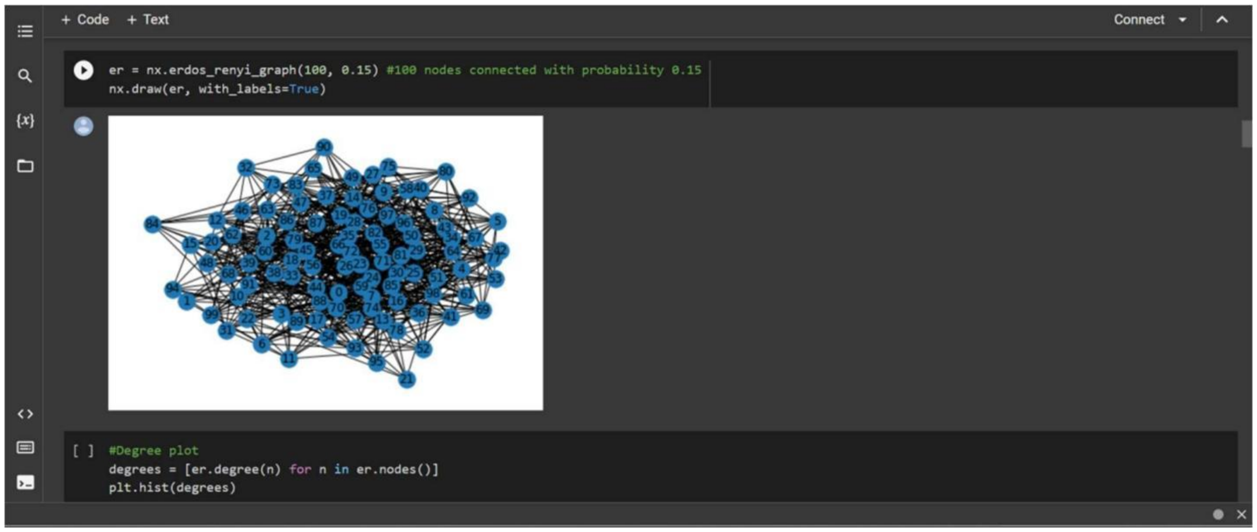
```
rt = nx.random_tree(20)
nx.draw(rt, with_labels=True)
```

```
[ ] bt = nx.balanced_tree(3,2)
    nx.draw(bt, with_labels=True)
```



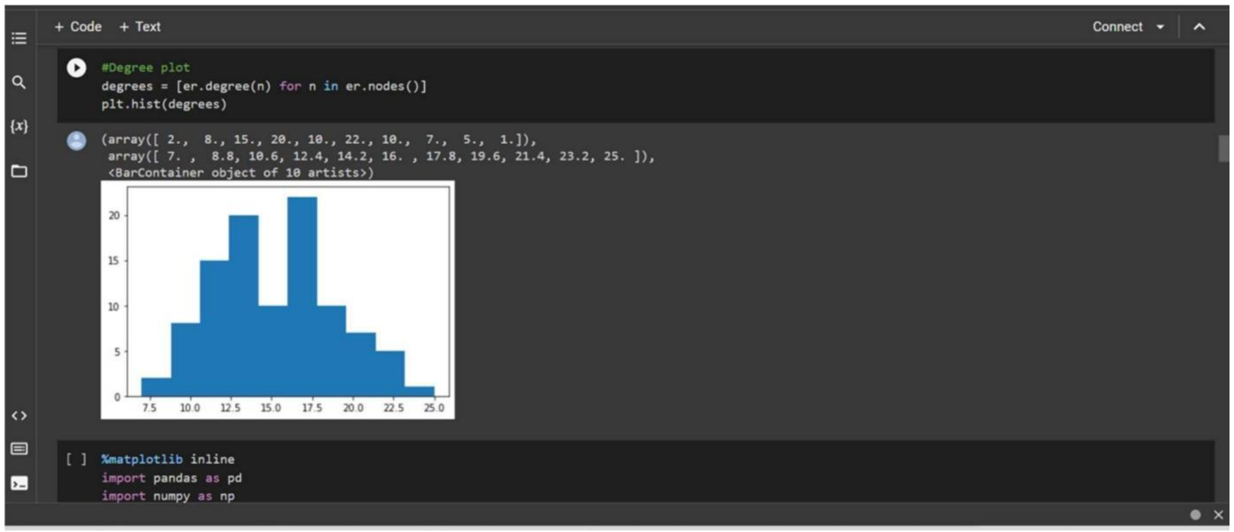
```
bt = nx.balanced_tree(3,2)
nx.draw(bt, with_labels=True)
```

```
er = nx.erdos_renyi_graph(100, 0.15) #100 nodes connected with probability 0.15
nx.draw(er, with_labels=True)
```



```
er = nx.erdos_renyi_graph(100, 0.15) #100 nodes connected with probability 0.15
nx.draw(er, with_labels=True)
```

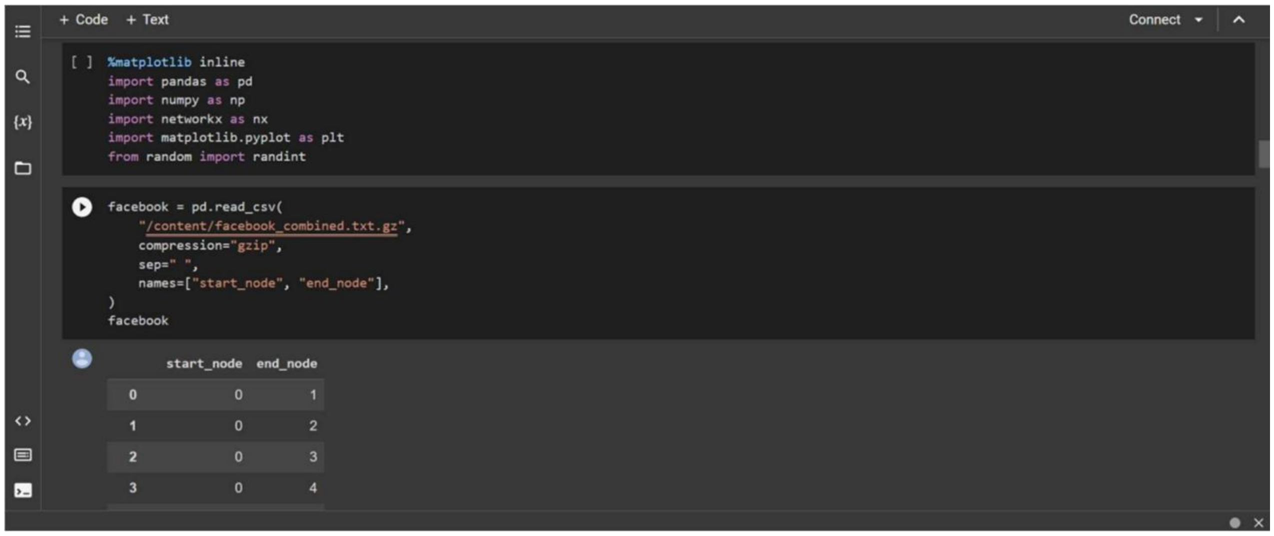
```
#Degree plot
degrees = [er.degree(n) for n in er.nodes()]
plt.hist(degrees)
```



```
#Degree plot
degrees = [er.degree(n) for n in er.nodes()]
plt.hist(degrees)
```

```
(array([ 2.,  8., 15., 20., 10., 22., 10.,  7.,  5.,  1.]),
 array([ 7. ,  8.8, 10.6, 12.4, 14.2, 16. , 17.8, 19.6, 21.4, 23.2, 25. ]),
 <BarContainer object of 10 artists>)
```

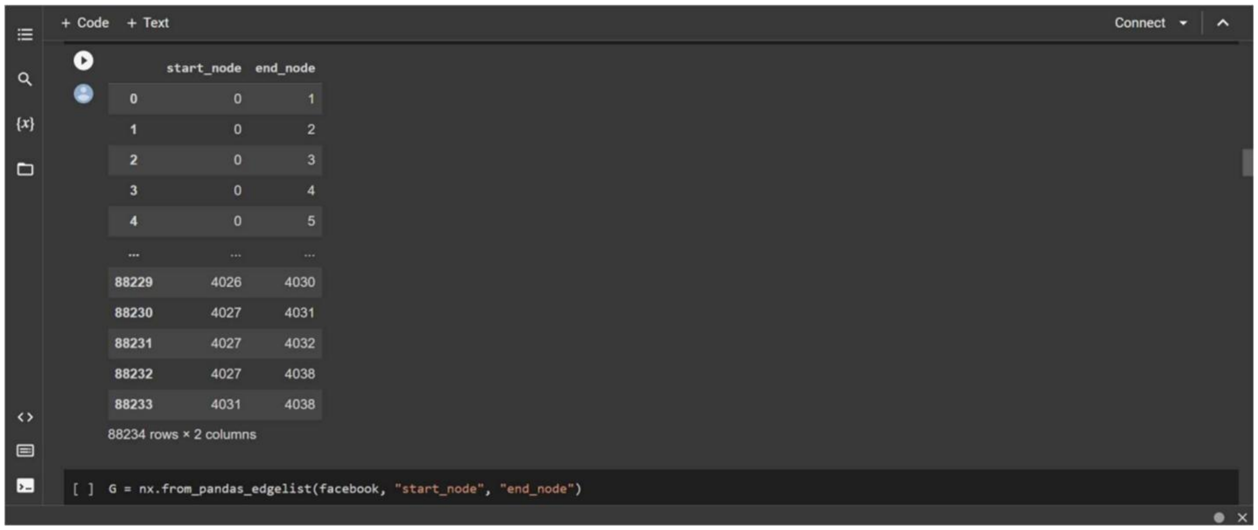
```
%matplotlib inline
import pandas as pd
import numpy as np
```



```
[ ] %matplotlib inline
    import pandas as pd
    import numpy as np
    import networkx as nx
    import matplotlib.pyplot as plt
    from random import randint
```

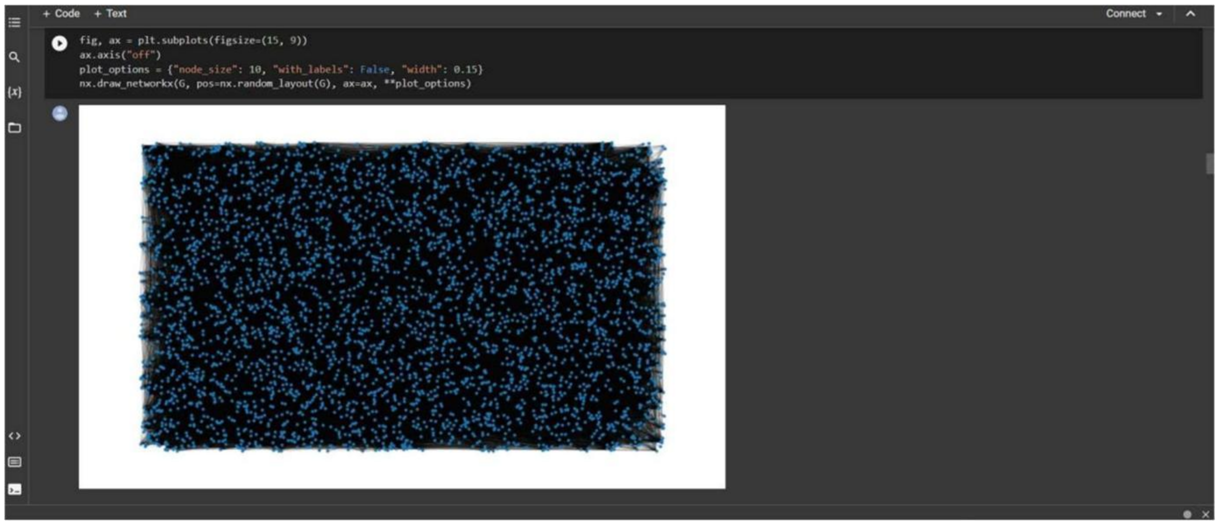
```
facebook = pd.read_csv(
    "/content/facebook_combined.txt.gz",
    compression="gzip",
    sep=" ",
    names=["start_node", "end_node"],
)
facebook
```

```
   start_node  end_node
0           0         1
1           0         2
2           0         3
3           0         4
```

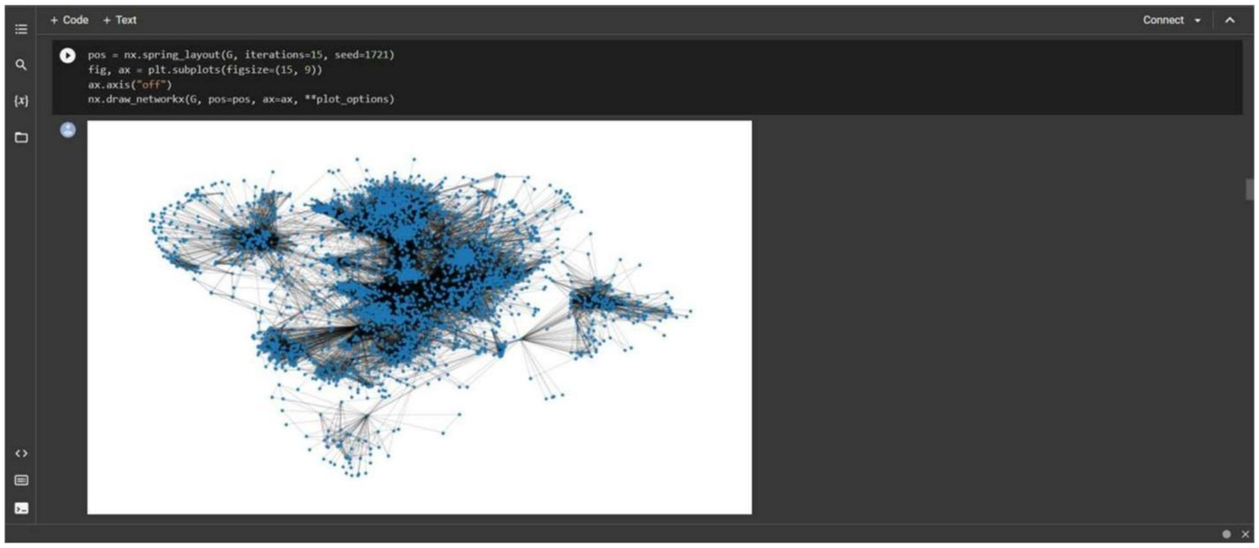


```
       start_node  end_node
0               0         1
1               0         2
2               0         3
3               0         4
4               0         5
...           ...       ...
88229        4026      4030
88230        4027      4031
88231        4027      4032
88232        4027      4038
88233        4031      4038

88234 rows × 2 columns
```

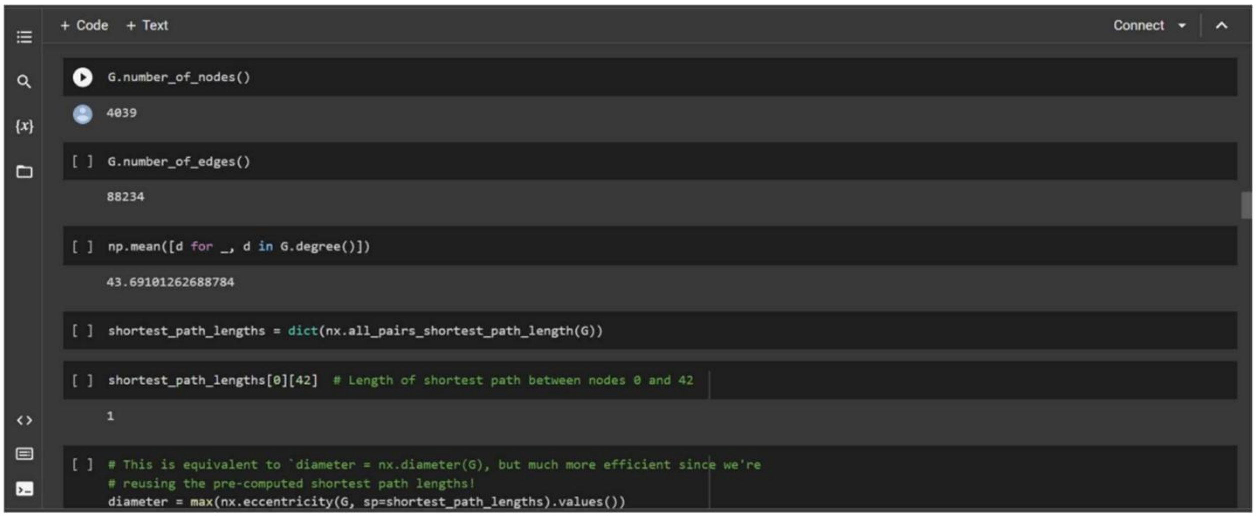
```
[ ] G = nx.from_pandas_edgelist(facebook, "start_node", "end_node")
```



```
fig, ax = plt.subplots(figsize=(15, 9))
ax.axis("off")
plot_options = {"node_size": 10, "with_labels": False, "width": 0.15}
nx.draw_networkx(G, pos=nx.random_layout(G), ax=ax, **plot_options)
```



```
pos = nx.spring_layout(G, iterations=15, seed=1721)
fig, ax = plt.subplots(figsize=(15, 9))
ax.axis("off")
nx.draw_networkx(G, pos=pos, ax=ax, **plot_options)
```



```
G.number_of_nodes()
```

```
4039
```

```
[ ] G.number_of_edges()
```

```
88234
```

```
[ ] np.mean([d for _, d in G.degree()])
```

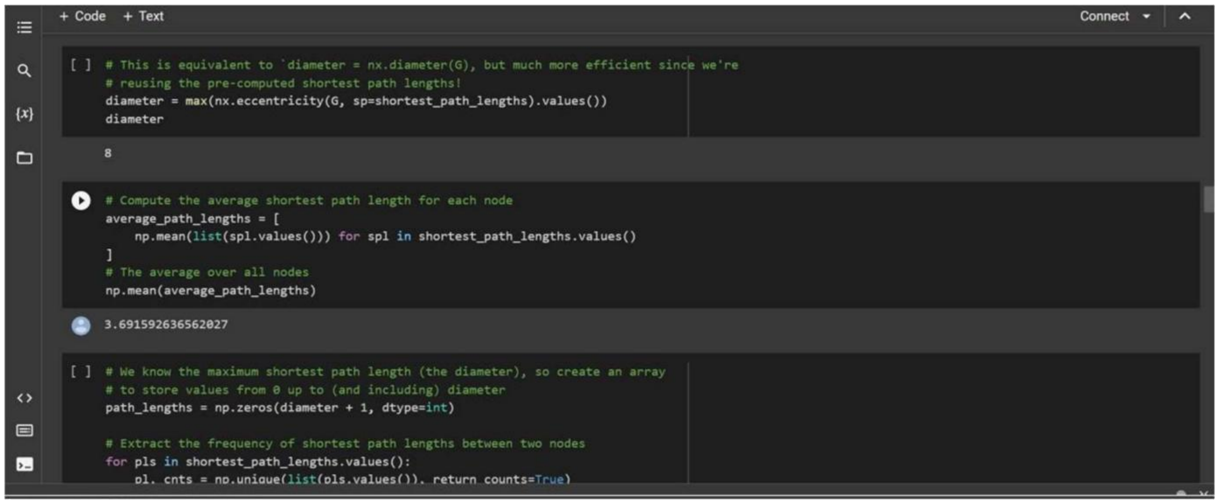
```
43.69101262688784
```

```
[ ] shortest_path_lengths = dict(nx.all_pairs_shortest_path_length(G))
```

```
[ ] shortest_path_lengths[0][42]  # Length of shortest path between nodes 0 and 42
```

```
1
```

```
[ ] # This is equivalent to `diameter = nx.diameter(G), but much more efficient since we're
    # reusing the pre-computed shortest path lengths!
    diameter = max(nx.eccentricity(G, sp=shortest_path_lengths).values())
```



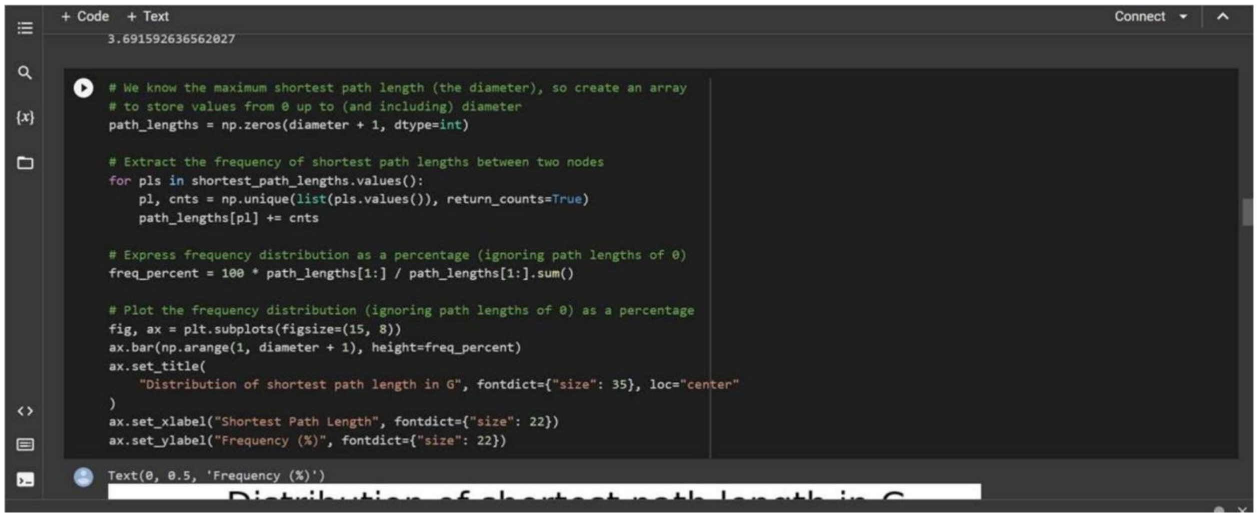
```
[ ] # This is equivalent to `diameter = nx.diameter(G), but much more efficient since we're
    # reusing the pre-computed shortest path lengths!
    diameter = max(nx.eccentricity(G, sp=shortest_path_lengths).values())
    diameter
```

```
8
```

```
# Compute the average shortest path length for each node
average_path_lengths = [
    np.mean(list(spl.values())) for spl in shortest_path_lengths.values()
]
# The average over all nodes
np.mean(average_path_lengths)
```

```
3.691592636562027
```

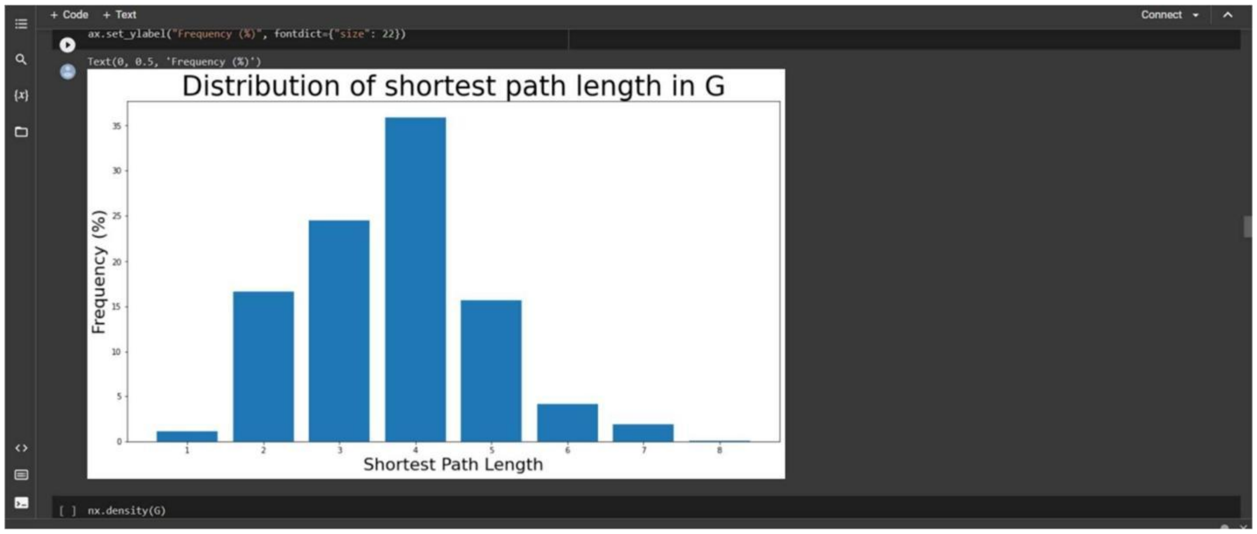
```
[ ] # We know the maximum shortest path length (the diameter), so create an array
    # to store values from 0 up to (and including) diameter
    path_lengths = np.zeros(diameter + 1, dtype=int)

    # Extract the frequency of shortest path lengths between two nodes
    for pls in shortest_path_lengths.values():
        pl, cnts = np.unique(list(pls.values()), return_counts=True)
```



```
3.691592636562027
```

```
# We know the maximum shortest path length (the diameter), so create an array
# to store values from 0 up to (and including) diameter
path_lengths = np.zeros(diameter + 1, dtype=int)

# Extract the frequency of shortest path lengths between two nodes
for pls in shortest_path_lengths.values():
    pl, cnts = np.unique(list(pls.values()), return_counts=True)
    path_lengths[pl] += cnts

# Express frequency distribution as a percentage (ignoring path lengths of 0)
freq_percent = 100 * path_lengths[1:] / path_lengths[1:].sum()

# Plot the frequency distribution (ignoring path lengths of 0) as a percentage
fig, ax = plt.subplots(figsize=(15, 8))
ax.bar(np.arange(1, diameter + 1), height=freq_percent)
ax.set_title(
    "Distribution of shortest path length in G", fontdict={"size": 35}, loc="center"
)
ax.set_xlabel("Shortest Path Length", fontdict={"size": 22})
ax.set_ylabel("Frequency (%)", fontdict={"size": 22})
```

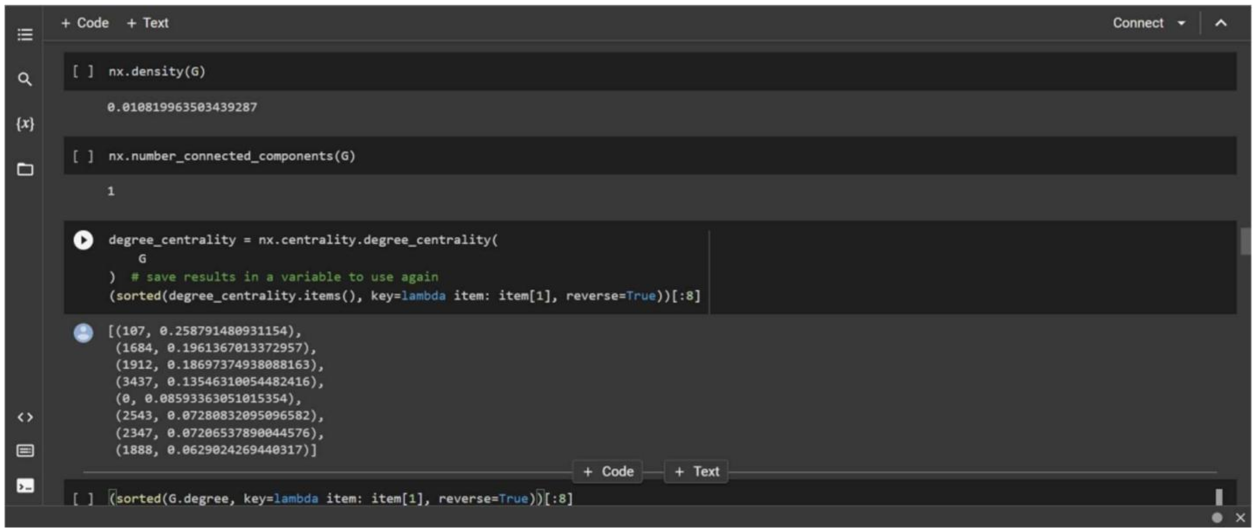
```
Text(0, 0.5, 'Frequency (%)')
```



```
ax.set_ylabel("Frequency (%)", fontdict={"size": 22})
```

```
Text(0, 0.5, 'Frequency (%)')
```

```
[ ] nx.density(G)
```



```
[ ] nx.density(G)
```

```
0.010819963503439287
```

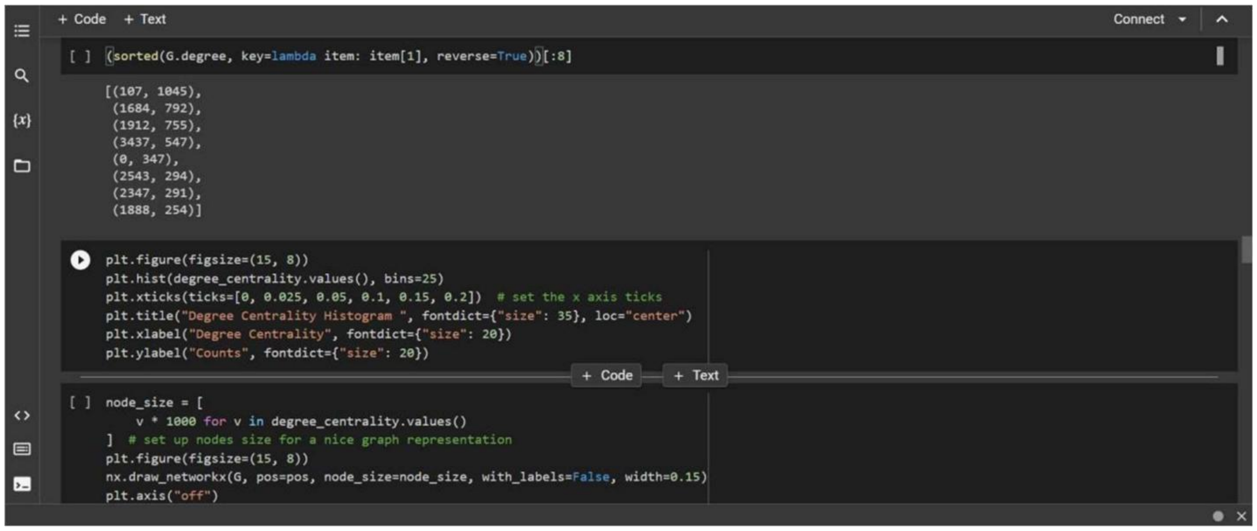
```
[ ] nx.number_connected_components(G)
```

```
1
```

```
degree_centrality = nx.centrality.degree_centrality(
    G
)  # save results in a variable to use again
(sorted(degree_centrality.items(), key=lambda item: item[1], reverse=True))[:8]
```

```
[(107, 0.258791480931154),
 (1684, 0.1961367013372957),
 (1912, 0.18697374938088163),
 (3437, 0.13546310054482416),
 (0, 0.08593363051015354),
 (2543, 0.07280832095096582),
 (2347, 0.07206537890044576),
 (1888, 0.0629024269440317)]
```

```
[ ] (sorted(G.degree, key=lambda item: item[1], reverse=True))[:8]
```

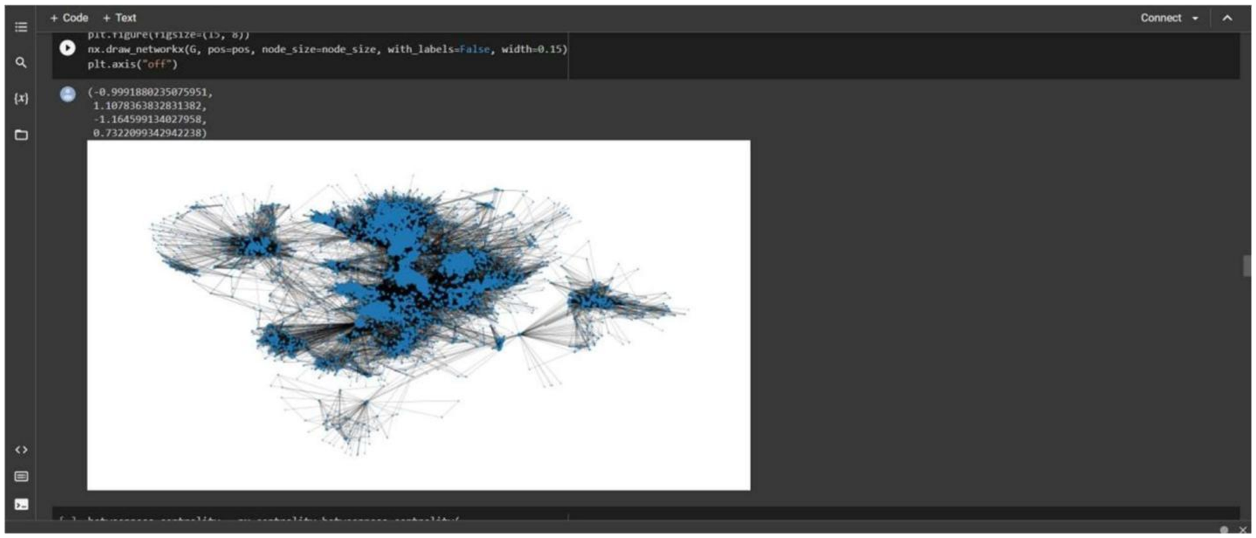


```
[ ] (sorted(G.degree, key=lambda item: item[1], reverse=True))[:8]
```

```
[(107, 1045),
 (1684, 792),
 (1912, 755),
 (3437, 547),
 (0, 347),
 (2543, 294),
 (2347, 291),
 (1888, 254)]
```

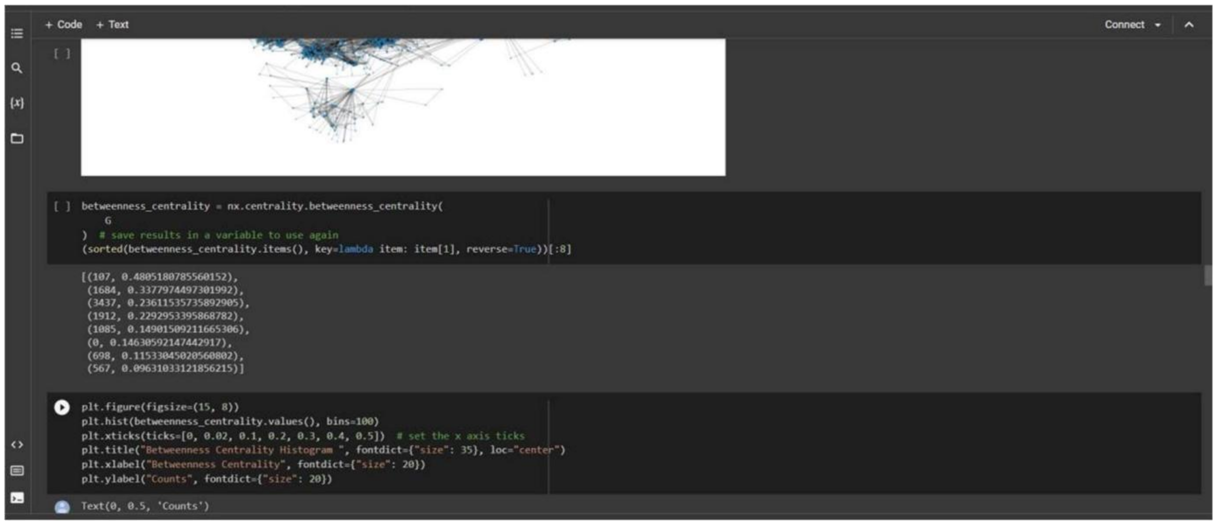
```
plt.figure(figsize=(15, 8))
plt.hist(degree_centrality.values(), bins=25)
plt.xticks(ticks=[0, 0.025, 0.05, 0.1, 0.15, 0.2])  # set the x axis ticks
plt.title("Degree Centrality Histogram ", fontdict={"size": 35}, loc="center")
plt.xlabel("Degree Centrality", fontdict={"size": 20})
plt.ylabel("Counts", fontdict={"size": 20})
```

```
[ ] node_size = [
        v * 1000 for v in degree_centrality.values()
    ]  # set up nodes size for a nice graph representation
    plt.figure(figsize=(15, 8))
    nx.draw_networkx(G, pos=pos, node_size=node_size, with_labels=False, width=0.15)
    plt.axis("off")
```



```
plt.figure(figsize=(15, 8))
nx.draw_networkx(G, pos=pos, node_size=node_size, with_labels=False, width=0.15)
plt.axis("off")
```

```
(-0.9991880235075951,
 1.1078363832831382,
 -1.164599134027958,
 0.7322099342942238)
```

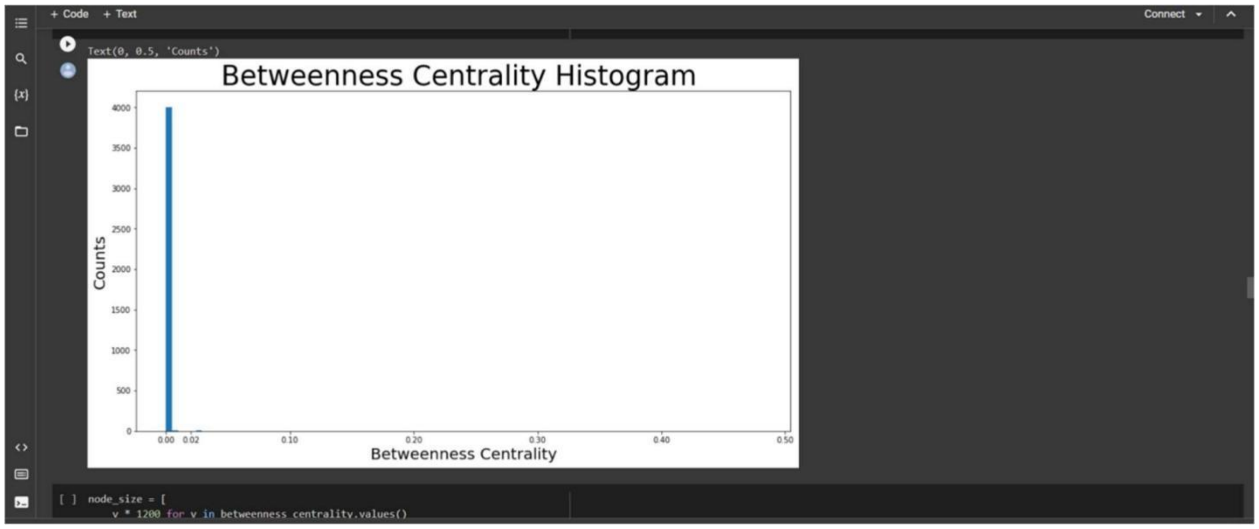


```
[ ] betweenness_centrality = nx.centrality.betweenness_centrality(
        G
    )  # save results in a variable to use again
    (sorted(betweenness_centrality.items(), key=lambda item: item[1], reverse=True))[:8]
```

```
[(107, 0.4805180785560152),
 (1684, 0.3377974497301992),
 (3437, 0.23611535735892905),
 (1912, 0.2292953395868782),
 (1085, 0.14901509211665306),
 (0, 0.14630592147442917),
 (698, 0.11533045020560802),
 (567, 0.09631033121856215)]
```

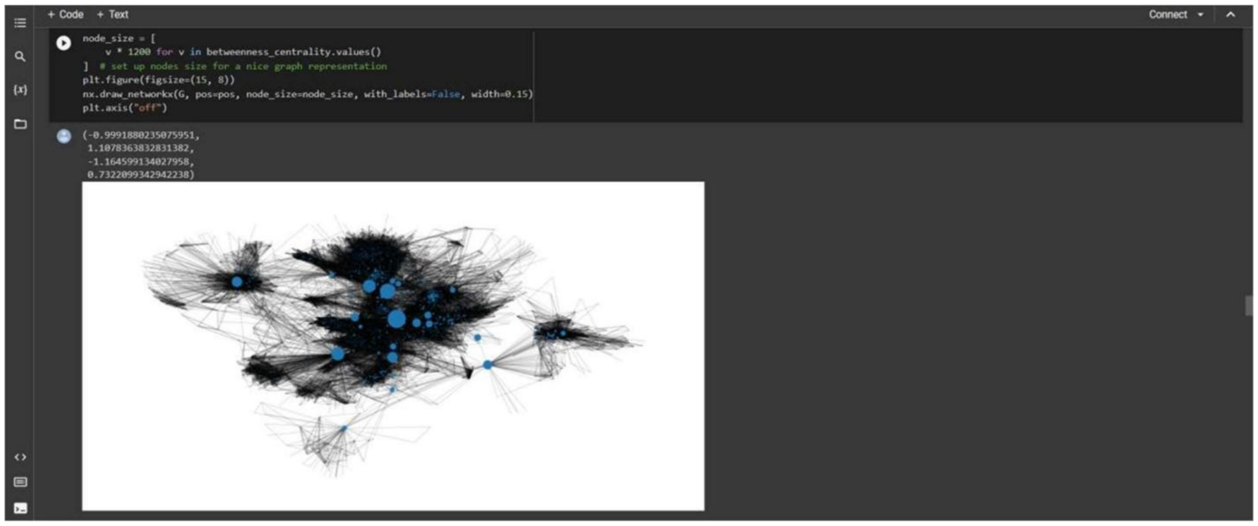
```
plt.figure(figsize=(15, 8))
plt.hist(betweenness_centrality.values(), bins=100)
plt.xticks(ticks=[0, 0.02, 0.1, 0.2, 0.3, 0.4, 0.5])  # set the x axis ticks
plt.title("Betweenness Centrality Histogram ", fontdict={"size": 35}, loc="center")
plt.xlabel("Betweenness Centrality", fontdict={"size": 20})
plt.ylabel("Counts", fontdict={"size": 20})
```

```
Text(0, 0.5, 'Counts')
```



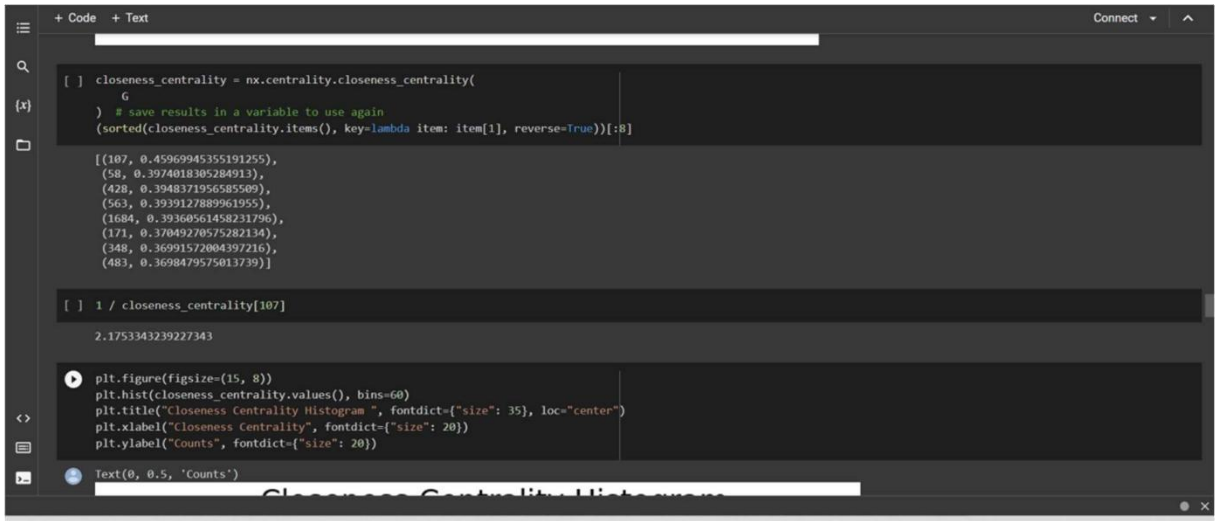
```
Text(0, 0.5, 'Counts')
```

```
[ ] node_size = [
        v * 1200 for v in betweenness_centrality.values()
```



```
node_size = [
    v * 1200 for v in betweenness_centrality.values()
]  # set up nodes size for a nice graph representation
plt.figure(figsize=(15, 8))
nx.draw_networkx(G, pos=pos, node_size=node_size, with_labels=False, width=0.15)
plt.axis("off")
```

```
(-0.9991880235075951,
 1.1078363832831382,
 -1.164599134027958,
 0.7322099342942238)
```



```
[ ] closeness_centrality = nx.centrality.closeness_centrality(
        G
    )  # save results in a variable to use again
    (sorted(closeness_centrality.items(), key=lambda item: item[1], reverse=True))[:8]
```

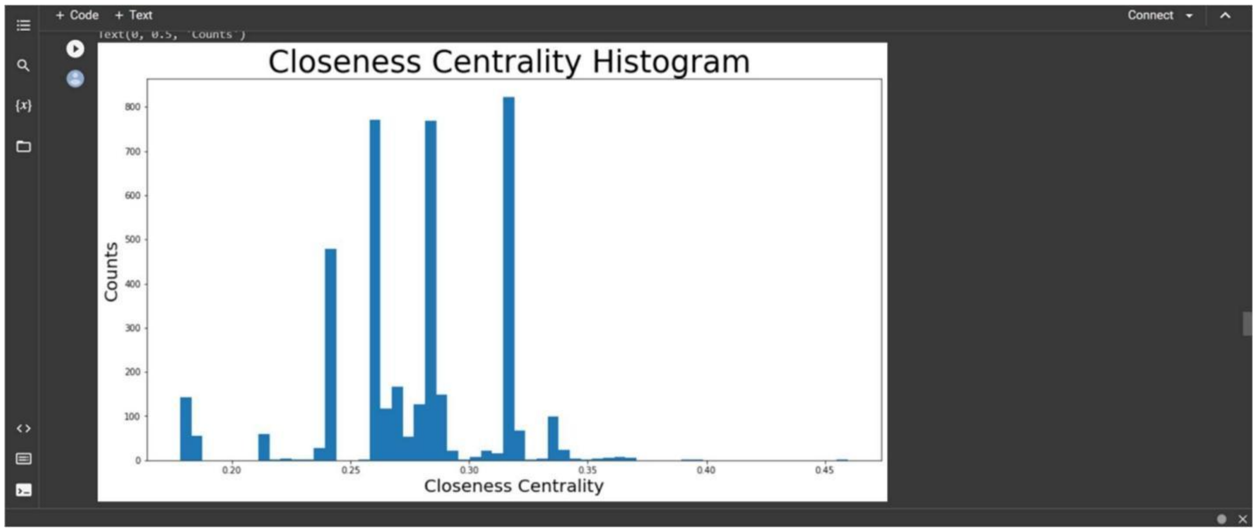
```
[(107, 0.45969945355191255),
 (58, 0.3974018305284913),
 (428, 0.3948371956585509),
 (563, 0.3939127889961955),
 (1684, 0.3936056145823179б),
 (171, 0.37049270575282134),
 (348, 0.36991572004397216),
 (483, 0.3698479575013739)]
```

```
[ ] 1 / closeness_centrality[107]
```

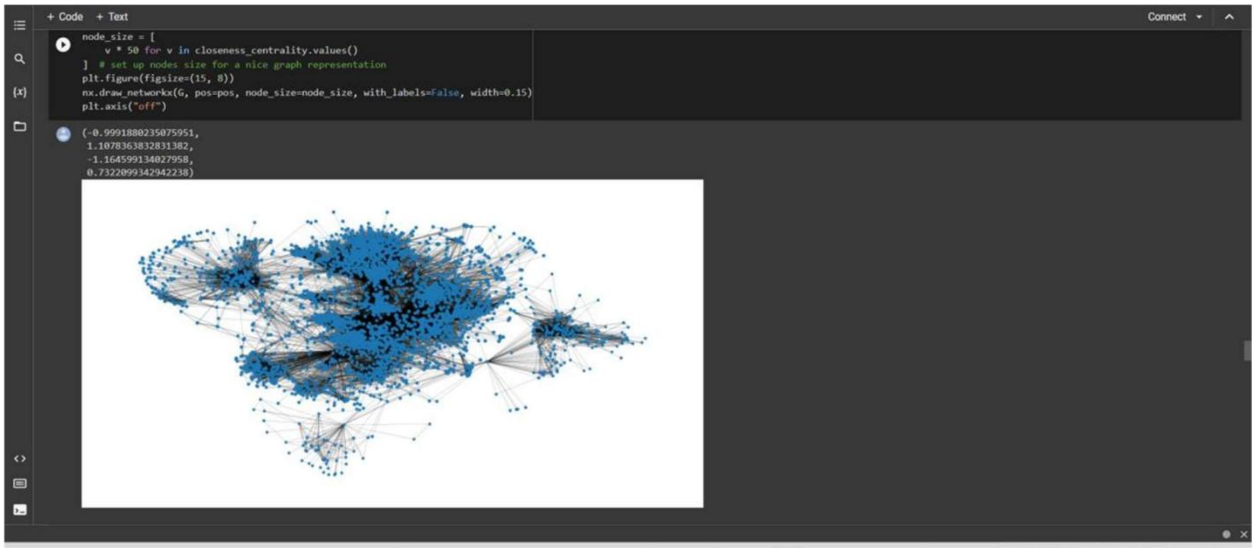
```
2.1753343239227343
```

```
plt.figure(figsize=(15, 8))
plt.hist(closeness_centrality.values(), bins=60)
plt.title("Closeness Centrality Histogram ", fontdict={"size": 35}, loc="center")
plt.xlabel("Closeness Centrality", fontdict={"size": 20})
plt.ylabel("Counts", fontdict={"size": 20})
```

```
Text(0, 0.5, 'Counts')
```

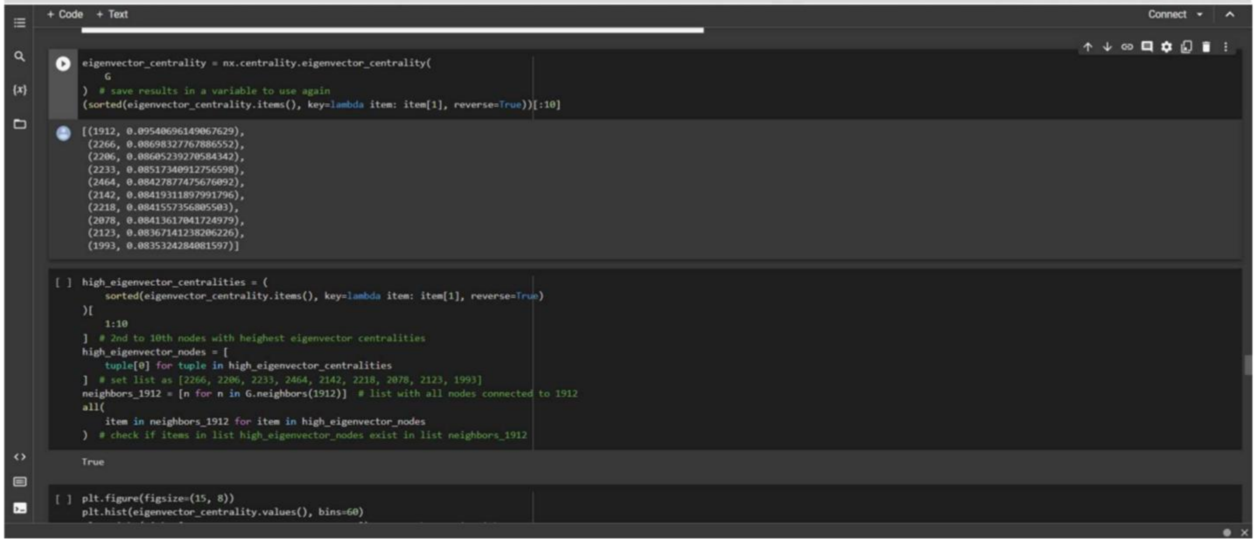


```
Text(0, 0.5, 'Counts')
```



```
node_size = [
    v * 50 for v in closeness_centrality.values()
]  # set up nodes size for a nice graph representation
plt.figure(figsize=(15, 8))
nx.draw_networkx(G, pos=pos, node_size=node_size, with_labels=False, width=0.15)
plt.axis("off")
```

```
(-0.9991880235075951,
 1.1078363832831382,
 -1.164599134027958,
 0.7322099342942238)
```



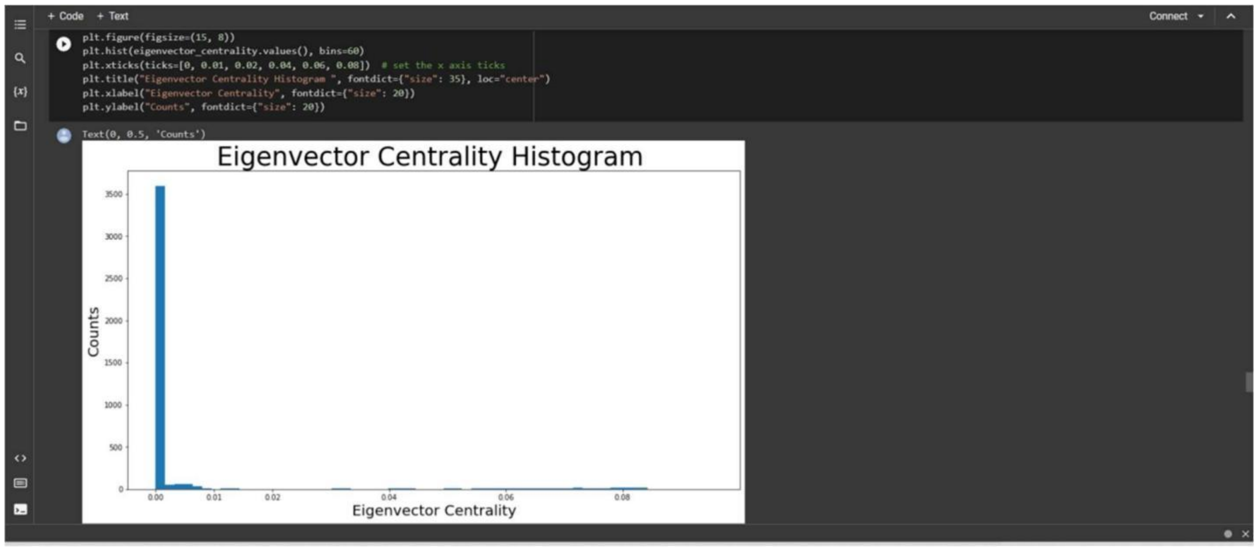
```
eigenvector_centrality = nx.centrality.eigenvector_centrality(
    G
)  # save results in a variable to use again
(sorted(eigenvector_centrality.items(), key=lambda item: item[1], reverse=True))[:10]
```

```
[(1912, 0.09540696149067629),
 (2266, 0.08698327767886552),
 (2206, 0.08605239270584342),
 (2233, 0.08517340912756598),
 (2464, 0.08427877475676092),
 (2142, 0.08419311897991796),
 (2218, 0.0841557356805503),
 (2078, 0.08413617041724979),
 (2123, 0.08367141238206226),
 (1993, 0.0835324284081597)]
```

```
high_eigenvector_centralities = (
    sorted(eigenvector_centrality.items(), key=lambda item: item[1], reverse=True)
)[
    1:10
]  # 2nd to 10th nodes with heighest eigenvector centralities
high_eigenvector_nodes = [
    tuple[0] for tuple in high_eigenvector_centralities
]  # set list as [2266, 2206, 2233, 2464, 2142, 2218, 2078, 2123, 1993]
neighbors_1912 = [n for n in G.neighbors(1912)]  # list with all nodes connected to 1912
all(
    item in neighbors_1912 for item in high_eigenvector_nodes
)  # check if items in list high_eigenvector_nodes exist in list neighbors_1912
```

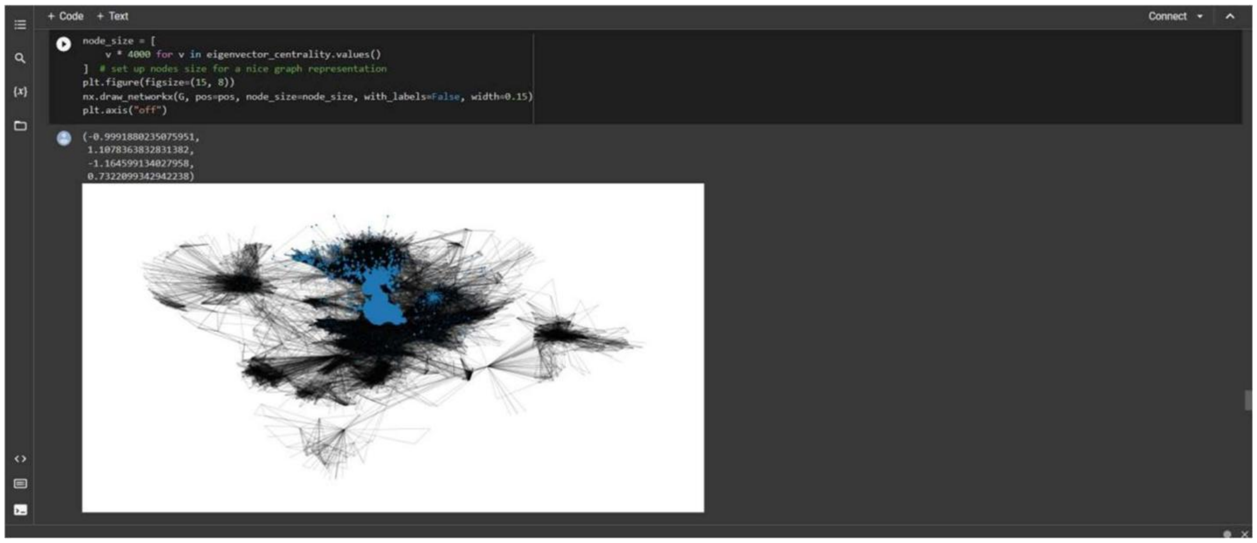
```
True
```

```
plt.figure(figsize=(15, 8))
plt.hist(eigenvector_centrality.values(), bins=60)
```



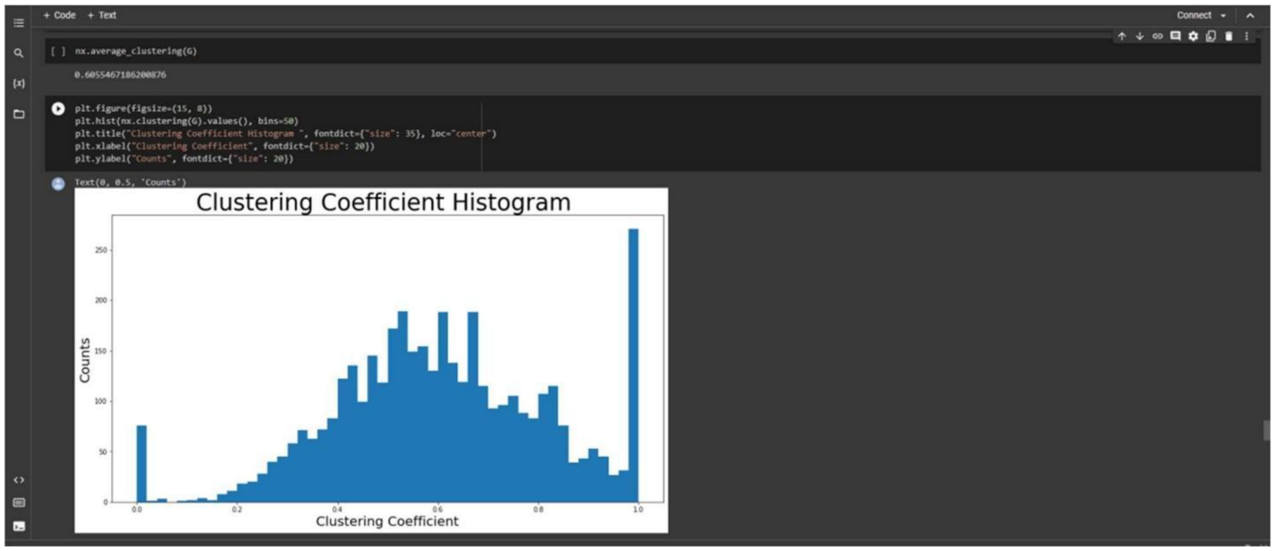
```
plt.figure(figsize=(15, 8))
plt.hist(eigenvector_centrality.values(), bins=60)
plt.xticks(ticks=[0, 0.01, 0.02, 0.04, 0.06, 0.08])  # set the x axis ticks
plt.title("Eigenvector Centrality Histogram ", fontdict={"size": 35}, loc="center")
plt.xlabel("Eigenvector Centrality", fontdict={"size": 20})
plt.ylabel("Counts", fontdict={"size": 20})
```

```
Text(0, 0.5, 'Counts')
```



```
node_size = [
    v * 4000 for v in eigenvector_centrality.values()
]  # set up nodes size for a nice graph representation
plt.figure(figsize=(15, 8))
nx.draw_networkx(G, pos=pos, node_size=node_size, with_labels=False, width=0.15)
plt.axis("off")
```

```
(-0.9991880235075951,
 1.1078363832831382,
 -1.164599134027958,
 0.7322099342942238)
```

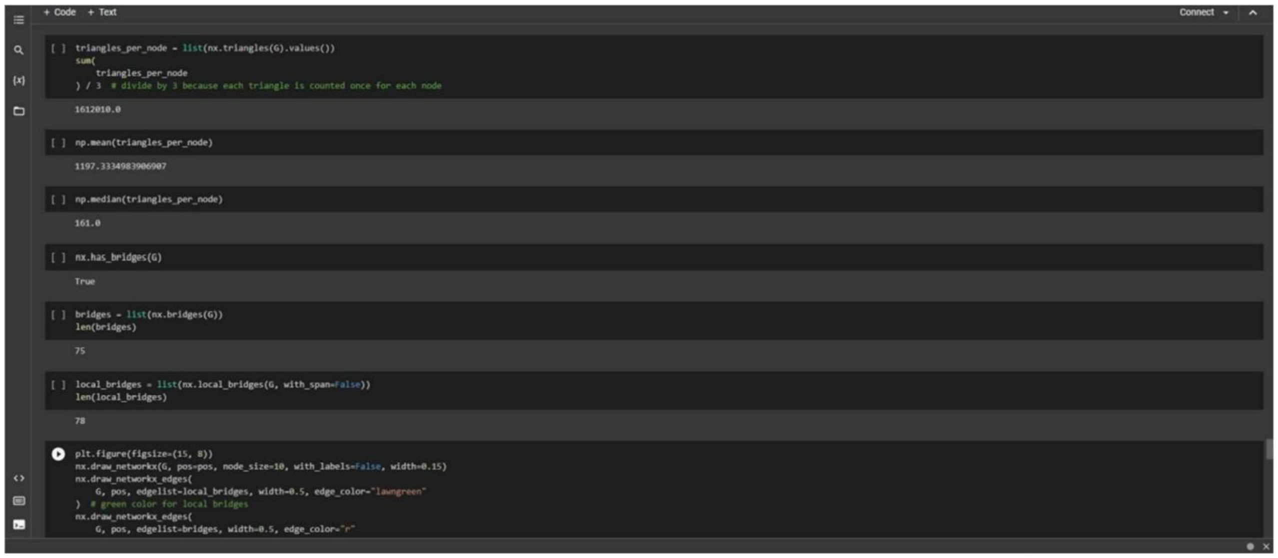


```
[ ] nx.average_clustering(G)
```

```
0.6055467186200876
```

```
plt.figure(figsize=(15, 8))
plt.hist(nx.clustering(G).values(), bins=50)
plt.title("Clustering Coefficient Histogram ", fontdict={"size": 35}, loc="center")
plt.xlabel("Clustering Coefficient", fontdict={"size": 20})
plt.ylabel("Counts", fontdict={"size": 20})
```

```
Text(0, 0.5, 'Counts')
```



```
triangles_per_node = list(nx.triangles(G).values())
sum(
    triangles_per_node
) / 3  # divide by 3 because each triangle is counted once for each node
```

```
1612010.0
```

```
np.mean(triangles_per_node)
```

```
1197.3334983906907
```

```
np.median(triangles_per_node)
```

```
161.0
```

```
nx.has_bridges(G)
```

```
True
```

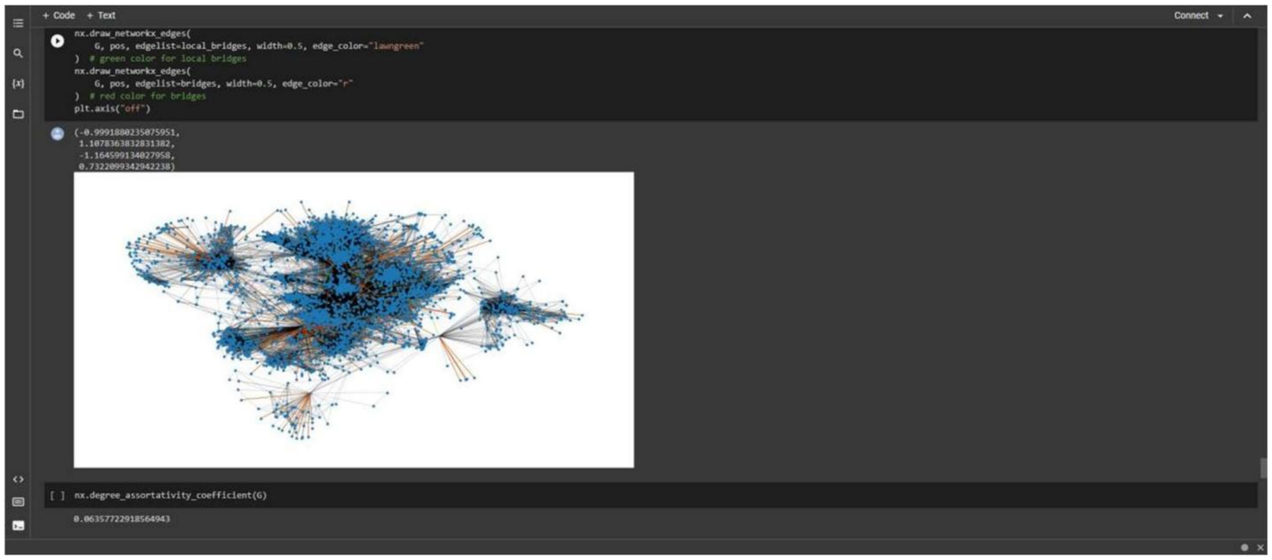
```
bridges = list(nx.bridges(G))
len(bridges)
```

```
75
```

```
local_bridges = list(nx.local_bridges(G, with_span=False))
len(local_bridges)
```

```
78
```

```
plt.figure(figsize=(15, 8))
nx.draw_networkx(G, pos=pos, node_size=10, with_labels=False, width=0.15)
nx.draw_networkx_edges(
    G, pos, edgelist=local_bridges, width=0.5, edge_color="lawngreen"
)  # green color for local bridges
nx.draw_networkx_edges(
    G, pos, edgelist=bridges, width=0.5, edge_color="r"
```



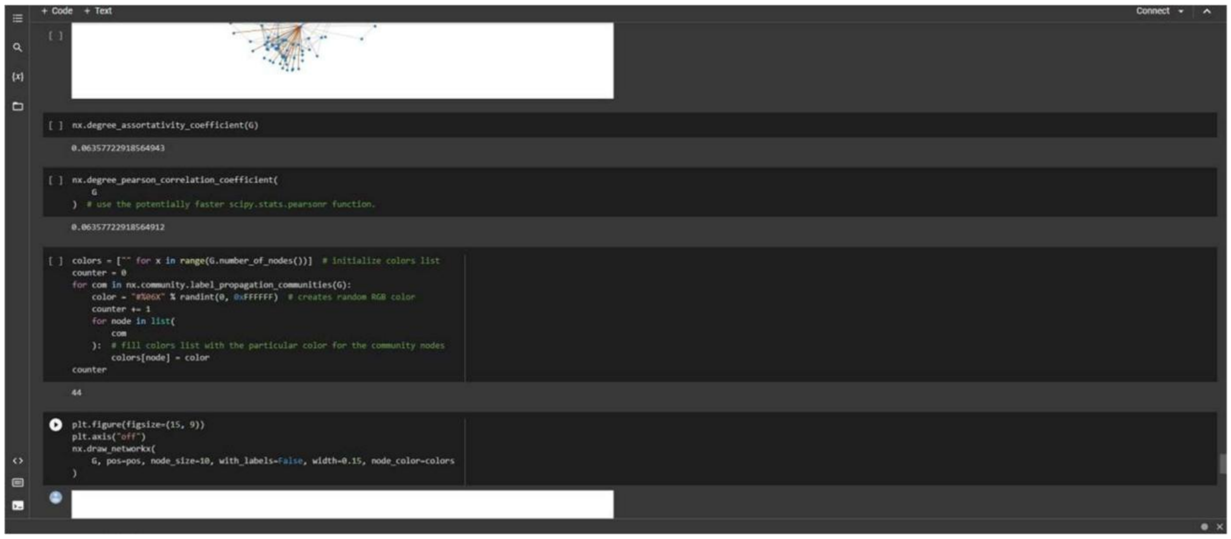
```
nx.draw_networkx_edges(
    G, pos, edgelist=local_bridges, width=0.5, edge_color="lawngreen"
)  # green color for local bridges
nx.draw_networkx_edges(
    G, pos, edgelist=bridges, width=0.5, edge_color="r"
```

```
)  # red color for bridges
plt.axis("off")
```

```
(-0.9991880235075951,
 1.1078363832831382,
 -1.164599134027958,
 0.7322099342942238)
```

```
nx.degree_assortativity_coefficient(G)
```

```
0.06357722918564943
```



```
nx.degree_assortativity_coefficient(G)
```

```
0.06357722918564943
```

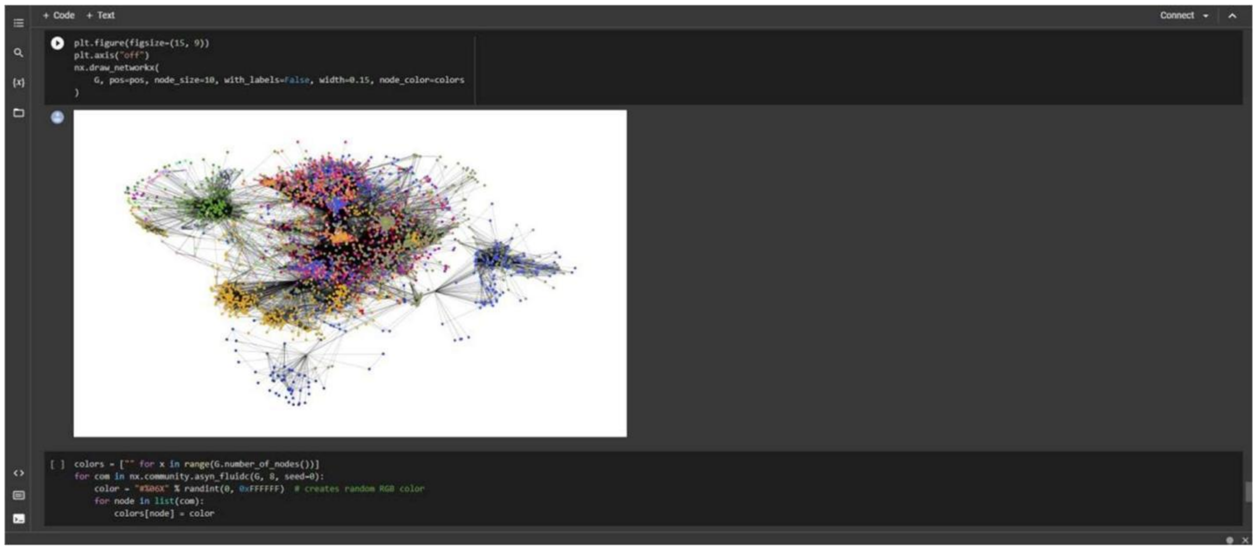
```
nx.degree_pearson_correlation_coefficient(
    G
)  # use the potentially faster scipy.stats.pearsonr function.
```

```
0.06357722918564912
```

```
colors = ["" for x in range(G.number_of_nodes())]  # initialize colors list
counter = 0
for com in nx.community.label_propagation_communities(G):
    color = "#%06X" % randint(0, 0xFFFFFF)  # creates random RGB color
    counter += 1
    for node in list(
        com
    ):  # fill colors list with the particular color for the community nodes
        colors[node] = color
counter
```

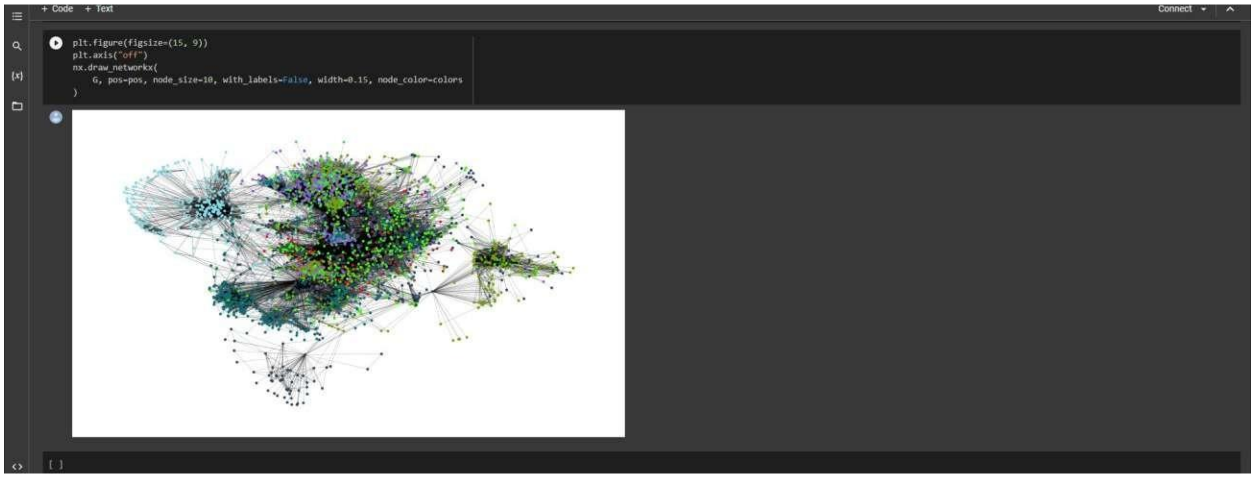
```
44
```

```
plt.figure(figsize=(15, 9))
plt.axis("off")
nx.draw_networkx(
    G, pos=pos, node_size=10, with_labels=False, width=0.15, node_color=colors
)
```



```
plt.figure(figsize=(15, 9))
plt.axis("off")
nx.draw_networkx(
    G, pos=pos, node_size=10, with_labels=False, width=0.15, node_color=colors
)
```

```
colors = ["" for x in range(G.number_of_nodes())]
for com in nx.community.asyn_fluidc(G, 8, seed=0):
    color = "#%06X" % randint(0, 0xFFFFFF)  # creates random RGB color
    for node in list(com):
        colors[node] = color
```



```
plt.figure(figsize=(15, 9))
plt.axis("off")
nx.draw_networkx(
    G, pos=pos, node_size=10, with_labels=False, width=0.15, node_color=colors
)
```

B.3 Observations and learning:

We get to know about Structure based social media analytics model for any business. (e.g., Structure Based Models -community detection, influence analysis. Also, observer betweenness, eigenvector, closeness centrality histogram, clustering coefficient histogram.

B.4 Conclusion:
In this experiment we able to collect, monitor, store and track social media data. Also, we develop Structure based social media analytics model for any business.

B.5 Question of Curiosity

Q1. What is network analysis? Explain its importance.
Answer: Network analysis, also known as social network analysis, is the study of relationships and connections between individuals, organizations, or other entities. It involves analyzing the structure of networks, including the patterns of relationships, the strength of connections, and the flow of information or resources.
In the context of social media analytics, network analysis is particularly important because it allows analysts to understand how people are connected and how they interact with each other online. By examining social media networks, analysts can identify key influencers, detect trends

and patterns in user behavior, and understand the flow of information within and between online communities.

Some of the key metrics and measures used in network analysis include degree centrality, which measures the number of connections a node (i.e., user) has within a network, and betweenness centrality, which measures the extent to which a node serves as a bridge or intermediary between other nodes in the network. Other measures include clustering coefficients, which measure the extent to which nodes in a network tend to form tightly connected clusters, and network density, which measures the overall level of connectivity within a network.

Overall, network analysis is a powerful tool for understanding the dynamics of social media networks, and can provide valuable insights into user behavior, trends, and patterns of communication. As social media continues to play an increasingly important role in our lives, the ability to analyze and understand these networks will only become more important for businesses, researchers, and policymakers alike.

Q2. What is structured analytics in social media?

Answer: Structured analytics in social media refers to the process of analyzing social media data in a structured and organized way to gain insights into user behavior, preferences, and trends. This approach involves using specialized software tools and techniques to collect, process, and analyze social media data from various sources, such as posts, comments, likes, and shares.

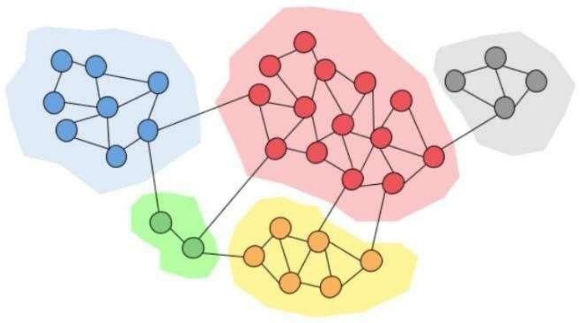
Structured analytics helps to identify patterns and trends in social media data, which can be used to develop marketing strategies, improve customer engagement, and optimize social media campaigns. It also helps to track key performance indicators (KPIs) and measure the impact of social media activities on business goals.

Some common examples of structured analytics in social media include sentiment analysis, network analysis, and social listening. Sentiment analysis involves analyzing the emotional tone of social media posts to gauge public opinion about a brand or product. Network analysis focuses on mapping and analyzing the connections between social media users to identify influencers and key opinion leaders. Social listening involves monitoring social media conversations to identify emerging trends and topics relevant to a brand or industry.

Overall, structured analytics in social media is an essential tool for businesses and organizations looking to leverage the vast amounts of data available on social media platforms to gain a competitive edge and improve customer engagement**.**

Q3. What is community detection in social networking? Enlist the different community detection techniques.

Answer: Community detection is one of the important tasks of social network analysis. It has significant practical importance for achieving cost-effective solutions for problems in the area of search engine optimization, spam detection, viral marketing, counter-terrorism, epidemic modeling, etc. Existing community detection techniques utilize the topological structure of the social network, but a proper combination of the available attribute data, which represents the properties of the participants or actors, and the structure data of the social network graph is promising for the detection of more accurate and meaningful communities.



There are several community detection techniques that are used in social networking such as:

- Modularity-based methods: These methods aim to maximize modularity which is a measure of how well a network can be partitioned into communities.
- Hierarchical clustering: This method involves grouping nodes into clusters based on their similarity.
- Spectral clustering: This method involves grouping nodes based on their similarity in a low-dimensional space.
- Label propagation: This method involves propagating labels through a network until convergence.
- Random walk-based methods: These methods involve simulating random walks on a network and using them to identify communities.

Q4. Explain social influence evaluation metrics.

Answer: Social influence evaluation metrics are used to measure the impact of social media influencers on their audience. These metrics help brands and marketers to determine the effectiveness of their social media campaigns and to identify the most influential people in a particular niche.

Here are some commonly used social influence evaluation metrics:

Reach: Reach is a metric that measures the number of people who have seen a particular social media post. It indicates the potential audience size of an influencer.

Engagement: Engagement measures the level of interaction between an influencer and their audience. It includes metrics such as likes, comments, shares, and retweets.

Impressions: Impressions are the number of times a social media post has been displayed to users. This metric is used to measure the potential exposure of an influencer's content.

Click-through rate (CTR): CTR is the number of clicks on a link in a social media post, divided by the number of impressions. It indicates the level of interest in a particular piece of content.

Conversion rate: Conversion rate measures the percentage of people who have taken a specific action, such as making a purchase, after being exposed to an influencer's content.

Brand awareness: Brand awareness measures the level of familiarity and recognition of a brand among an influencer's audience.

Share of voice: Share of voice is the percentage of online conversations about a particular brand or product that an influencer is responsible for.

Overall, these metrics help to evaluate the effectiveness of social media influencer campaigns, identify areas for improvement, and make data-driven decisions about marketing strategies.

Q5. Explain in detail the different network measures to describe the structure of a network.
Answer: Network measures are quantitative metrics that can be used to describe the structure of a network. These measures can help us understand various aspects of the network, such as its size, connectivity, centrality, and resilience. In this answer, we will discuss some of the most commonly used network measures.

1. **Degree:** The degree of a node in a network is the number of edges connected to it. In a directed network, we can distinguish between in-degree (number of incoming edges) and out-degree (number of outgoing edges). The degree distribution is the probability distribution of the degrees of all nodes in the network.
2. **Average Path Length**: The average path length is the average number of steps along the shortest paths for all possible pairs of nodes in the network. It gives an idea of how efficiently information can be transmitted across the network.
3. **Clustering Coefficient:** The clustering coefficient of a node is the proportion of its neighbors that are also neighbors of each other. The clustering coefficient of the network is the average of the clustering coefficients of all nodes. It measures the degree to which nodes in a network tend to cluster together.
4. **Betweenness Centrality:** The betweenness centrality of a node is the number of shortest paths between any two nodes in the network that pass through that node. It measures the importance of a node in the network's communication structure, as nodes with high betweenness centrality act as bridges between different parts of the network.
5. **Eigenvector Centrality:** Eigenvector centrality measures the influence of a node in the network by taking into account the centrality of its neighbors. A node with high eigenvector centrality is connected to other nodes with high centrality, making it important in the network.
6. **Degree Distribution**: The degree distribution is the probability distribution of the degrees of all nodes in the network. It tells us how many nodes have a certain degree and how common those degrees are in the network.
7. **Network Density:** The network density is the ratio of the number of edges in the network to the total number of possible edges. It gives an idea of how interconnected the network is.
8. **Modularity:** Modularity is a measure of how well the network can be divided into modules or communities. It is based on the idea that nodes within a module should be more densely connected to each other than to nodes in other modules.
9. **Assortativity:** Assortativity is a measure of how nodes in a network tend to connect to other nodes with similar or dissimilar properties. It can be measured by the degree Assortativity coefficient, which measures the correlation between the degrees of nodes at either end of an edge.

These are just a few of the many network measures that can be used to describe the structure of a network. The choice of measure depends on the specific research question and the characteristics of the network being studied.