In [479…

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

Networkx

Networkx is a Python library people commonly use when dealing with graphs. Please follow the instructions to install the package: https://networkx.org/documentation/stable/install.html. A tutorial for quick-start is also available on their webpage: https://networkx.org/documentation/stable/tutorial.html.

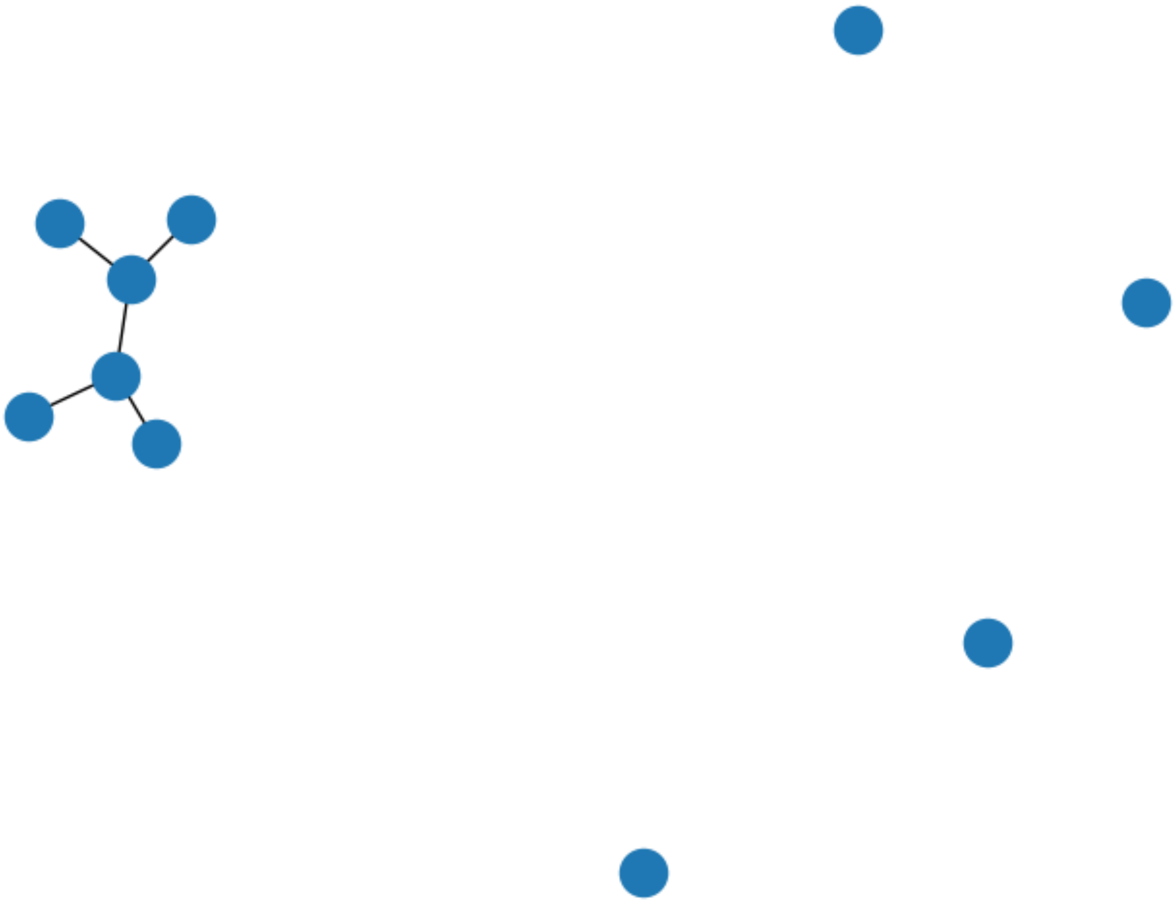
$G(n, p)$

This notebook aims to empirically investigate the properties of Erdős-Rényi graphs, specifically the $G(n, p)$ model, and compare them with real-world networks. Networkx provides a convenient way to generate random graphs using built-in functions for various commonly used graph models. To create an Erdős-Rényi graph, we can use the function `nx.erdos_renyi_graph(n, p)`, where `n` is the number of nodes and `p` is the probability of an edge between two nodes. The function returns an `nx.Graph` object containing the nodes and edges of the generated graph.

In [480…

```
# Example: generate a graph from G(10, 0.1)
G_test = nx.erdos_renyi_graph(10, 0.1)
# Check how many nodes and edges it has.
print(G_test.number_of_nodes(), G_test.number_of_edges())
# Plot the graph
nx.draw(G_test)
```

```
10 5
```



Statistics of $G(n, p)$ (20 pts)

In the upcoming code cell, you will be requested to generate graphs from $G(n, p)$ model with n set to 100, using different values of p ranging from 0.001 to 0.081, with a small step size such as 0.005. For each value of p, you should generate 10 graph samples from the model and report the average of the following graph statistics with error bars for std. We also provide a brief explanation on how to compute these statistics using networkx functions:

- Number of edges: To obtain the number of edges in a given nx.graph object G, you can utilize the function `G.number_of_edges()`.
- Number of triangles: Using `nx.triangles(G)`, a dictionary of (node id, number of triangles participated) key-value pairs is returned. To calculate the total number of triangles in G, simply sum all the values in the dictionary and divide the result by 3 (since each triangle is counted three times in the dictionary).
- Number of isolated nodes: If a node has degree 0, then it is an isolated node. The function G.degree is a map-like object consisting of (node id, node degree) pairs. To count the number of isolated nodes, iterate through `G.degree` and count the number of 0s in the values.
- Number of connected components: In graph theory, a connected component is a set of vertices in a graph that are linked to each other by paths. Using nx.connected_components(G), a node list generator is returned, which yields one component at a time. To get the total number of connected components, you can use `len(list(nx.connected_components(G)))`.

For each graph statistic, generate a plot where the x-axis represents the values of p, and the y-axis represents the average of the statistic.

In [481…

```
# Write your code below

p = 0.001
step_size = 0.005

g_edges = []
edges = []
e_error = []

g_triangles = []
triangles = []
```

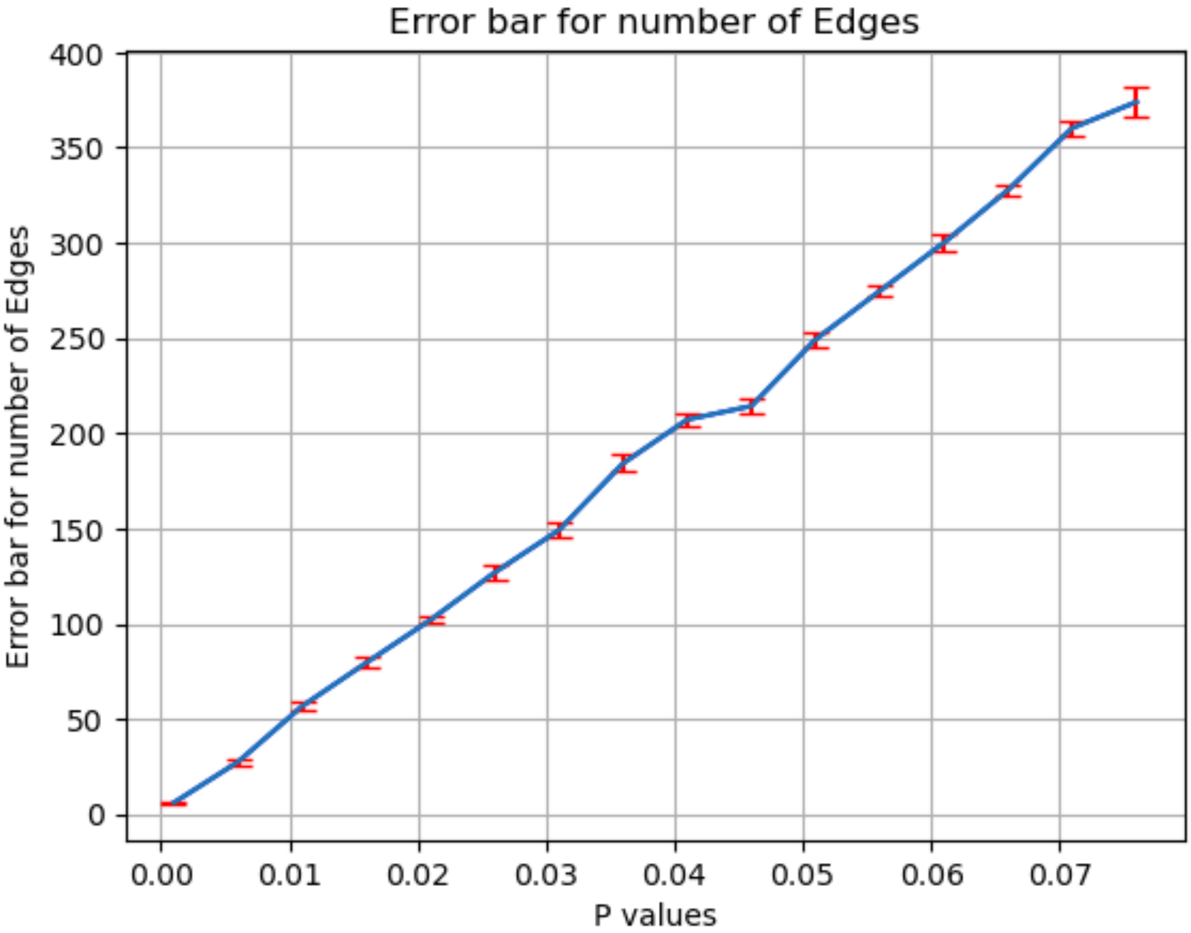
```
t_error = []

g_isolatedNodes = []
isolatedNodes = []
i_error = []

g_connectedComponents = []
connectedComponents = []
c_error = []

distinct_p = []

while p < 0.081:
    distinct_p.append(p)

    for x in range(10):
        G = nx.erdos_renyi_graph(100, p)

        edges.append(G.number_of_edges())

        tri = nx.triangles(G)
        tri_sum = sum(tri.values())/3
        triangles.append(tri_sum)

        sum_degree = 0
        degrees = [val for (node, val) in G.degree]
        for x in degrees:
            if x == 0:
                sum_degree += 1
        isolatedNodes.append(sum_degree)

        connectedComponents.append(len(list(nx.connected_components(G))))

    e = sum(edges)/len(edges)
    t = sum(triangles)/len(triangles)
    i = sum(isolatedNodes)/len(isolatedNodes)
    c = sum(connectedComponents)/len(connectedComponents)

    deviation = statistics.stdev(edges)
    e_error.append(deviation/np.sqrt(10))

    deviation = statistics.stdev(triangles)
    t_error.append(deviation/np.sqrt(10))

    deviation = statistics.stdev(isolatedNodes)
    i_error.append(deviation/np.sqrt(10))

    deviation = statistics.stdev(connectedComponents)
    c_error.append(deviation/np.sqrt(10))


    g_edges.append(e)
    g_triangles.append(t)
    g_isolatedNodes.append(i)
    g_connectedComponents.append(c)

    edges = []
    triangles = []
    isolatedNodes = []
    connectedComponents = []
    p += step_size
```

In [482…

```
# Generate plot with error bars for number of edges.

plt.plot(distinct_p, g_edges, color = "blue")
plt.grid()
plt.xlabel("P values")
plt.ylabel("Error bar for number of Edges")
plt.title("Error bar for number of Edges")
plt.errorbar(distinct_p,g_edges,yerr = e_error, ecolor = "red", capsize = 4)
```

Out[482]:

```
<ErrorbarContainer object of 3 artists>
```

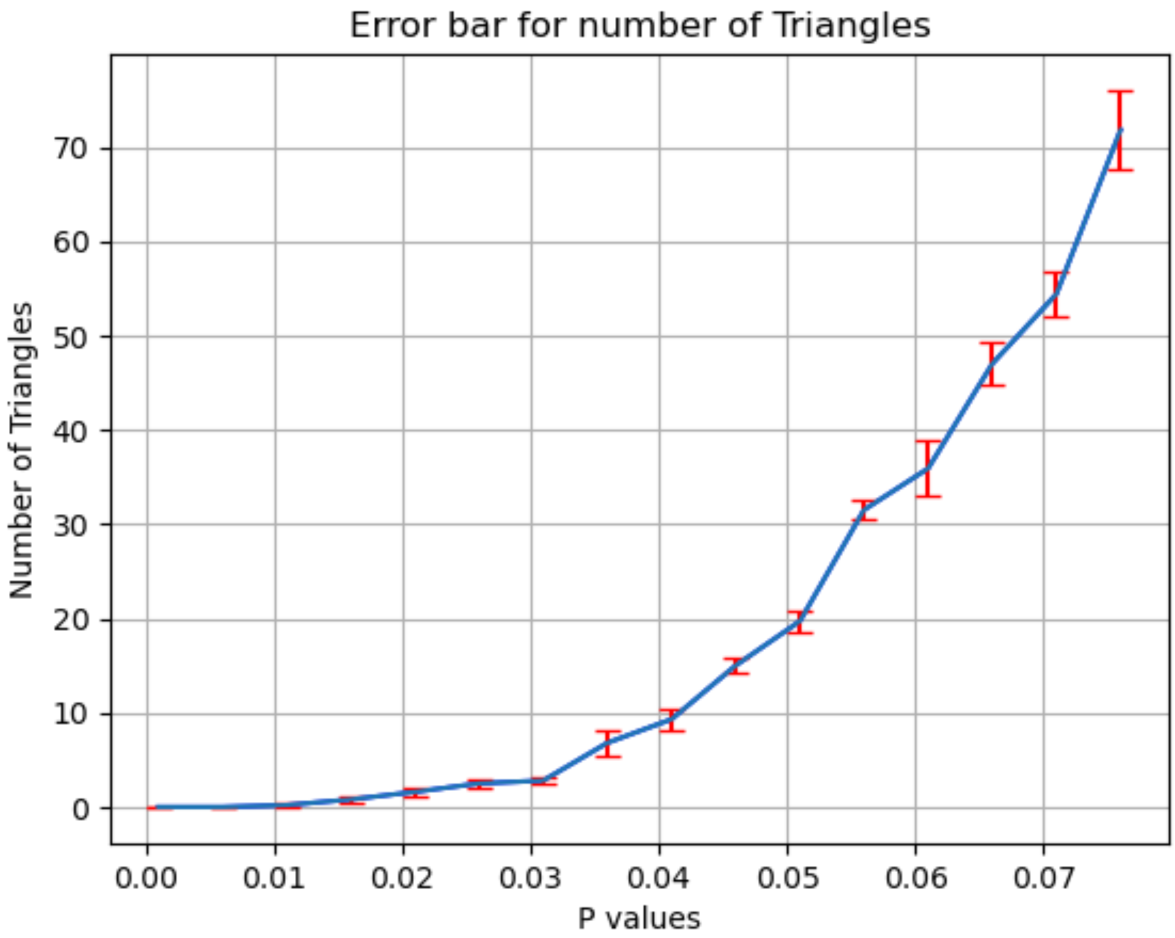


In [483…

```
# Generate plot with error bars for number of triangles.

plt.plot(distinct_p, g_triangles_for_part_one, color = "blue")
plt.grid()
plt.xlabel("P values")
plt.ylabel("Number of Triangles")
plt.title("Error bar for number of Triangles")
plt.errorbar(distinct_p,g_triangles_for_part_one,yerr = t_error_for_part_one
```

Out[483]:

```
<ErrorbarContainer object of 3 artists>
```

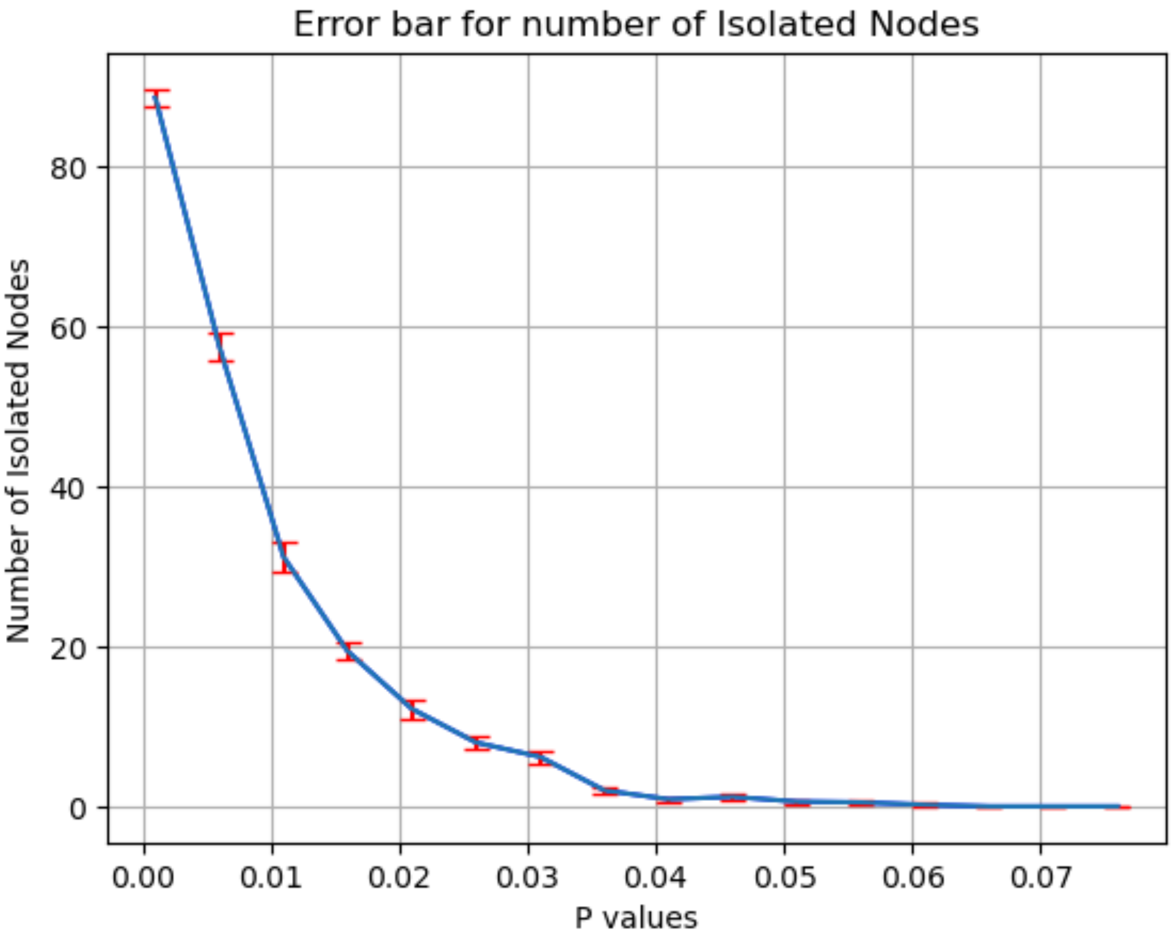


In [484…

```
# Generate plot with error bars for number of isolated nodes.

plt.plot(distinct_p, g_isolatedNodes, color = "blue")
plt.grid()
plt.xlabel("P values")
plt.ylabel("Number of Isolated Nodes")
plt.title("Error bar for number of Isolated Nodes")
plt.errorbar(distinct_p,g_isolatedNodes,yerr = i_error, ecolor = "red", caps
```

Out[484]:

```
<ErrorbarContainer object of 3 artists>
```

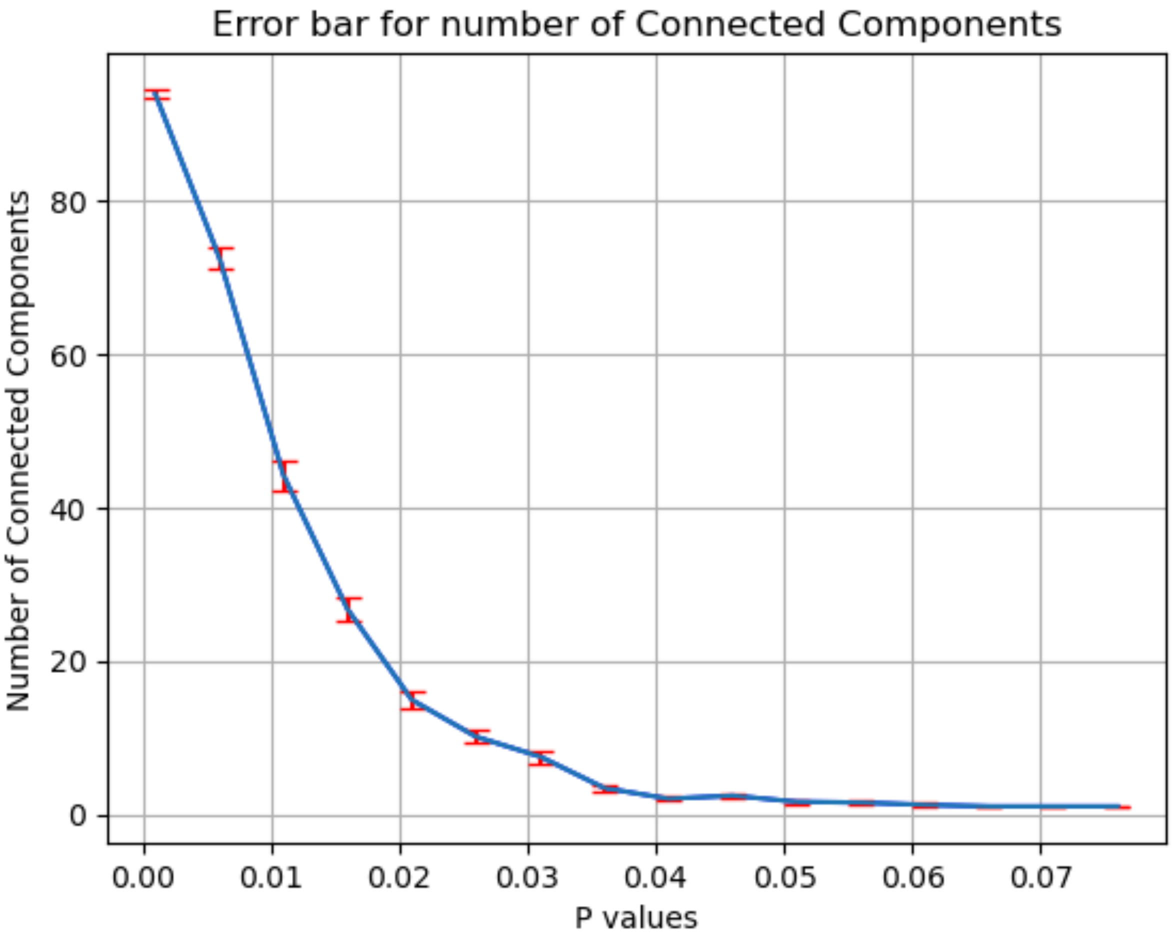


In [485…

```
# Generate plot with error bars for number of connected components.

plt.plot(distinct_p, g_connectedComponents, color = "blue")
plt.grid()
plt.xlabel("P values")
plt.ylabel("Number of Connected Components")
plt.title("Error bar for number of Connected Components")
plt.errorbar(distinct_p,g_connectedComponents,yerr = c_error, ecolor = "red"
```

Out[485]:

```
<ErrorbarContainer object of 3 artists>
```



Real world network

Graph statistics (10pts)

In the same folder we provide a real-world social network dataset saved as edgelist format named "fb-pages-food.edges". Each line in this file has the format of "node1,node2" that represents an edge connecting node1 and node2. An edgelist file can be loaded as nx.Graph directly using nx.read_edgelist() like the following.

In [486…

```
G = nx.read_edgelist("fb-pages-food.edges", delimiter=',')
```

How many nodes and edges are there in this graph? Also report the number of triangles, number of isolated nodes, and number of connected components in this graph.

In [487…

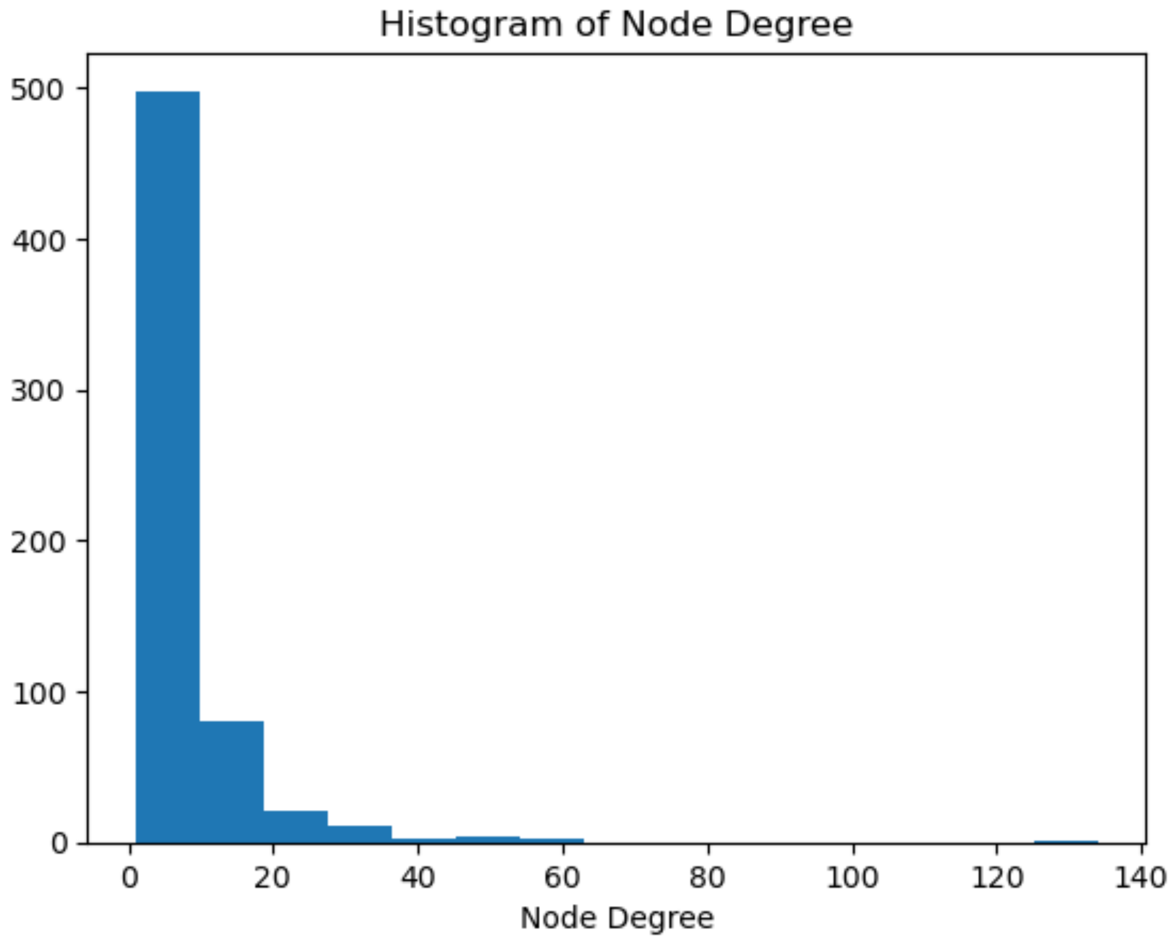
```
# Report the graph statistics.

tri = nx.triangles(G)
tri_sum = sum(tri.values())/ 3
tri_sum = int(tri_sum)

sum_degree = 0
degrees_for_real_data = [val for (node, val) in G.degree]
for x in degrees_for_real_data:
    if x == 0:
        sum_degree += 1

connectedComponents = len(list(nx.connected_components(G)))





print("There are " + str(G.number_of_nodes()) + " nodes in this graph.")
print("There are " + str(G.number_of_edges()) + " edges in this graph.")
print("There are " + str(tri_sum) + " triangles in this graph." )
print("There are " + str(sum_degree) + " isolated nodes in this graph.")
print("There are " + str(connectedComponents) + " connected components in th
```

```
There are 620 nodes in this graph.
There are 2102 edges in this graph.
There are 2935 triangles in this graph.
There are 0 isolated nodes in this graph.
There are 1 connected components in this graph.
```

In the next cell, plot a histogram of the node degrees in this graph with bins=15.

In [488…

```
fig, ax = plt.subplots()
ax.hist(degrees_for_real_data, bins = 15)
ax.set_title("Histogram of Node Degree")
ax.set_xlabel("Node Degree")
plt.show()
```



Try to fit the data with G(n,p) (10pts)

If we want to fit the graph with G(n,p) model, i.e., find a G(n,p) model whose expected number of edges equals to number of edge in this graph, How should we set p to get m edges in expectation?

In [489…

```
# caluclate and print the value of p.

max_edges = G.number_of_nodes()*(G.number_of_nodes()-1)/2
actual_edges = G.number_of_edges()
p = actual_edges/max_edges
print("The value of p is " + str(p))
```

```
The value of p is 0.0109541925061233
```

Now use the p value you calculated and sample 10 graphs from G(n,p). Report the same graph statistics in average.

In [490…

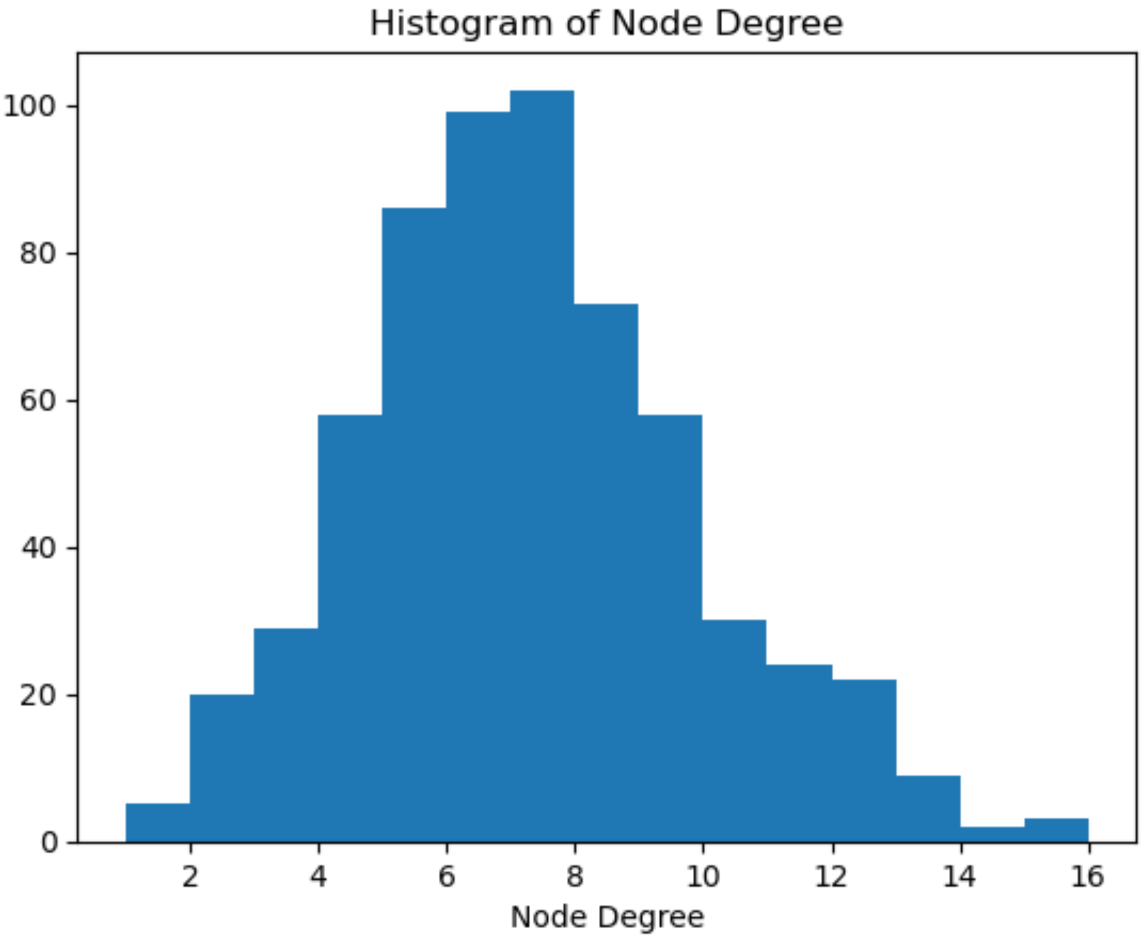
```
edges = []
triangles = []
isolated_nodes = []
connected_components = []

for x in range(10):
    G = nx.erdos_renyi_graph(G.number_of_nodes(),p)

    edges.append(G.number_of_edges())

    tri = nx.triangles(G)
    tri_sum = sum(tri.values())/ 3
    tri_sum = int(tri_sum)
    triangles.append(tri_sum)

    sum_degree = 0
    degrees = [val for (node, val) in G.degree]
    for x in degrees:
        if x == 0:
            sum_degree += 1
    isolated_nodes.append(sum_degree)

    connected_components.append(len(list(nx.connected_components(G))))

one = sum(edges)/len(edges)
two = sum(triangles)/len(triangles)
three = sum(isolated_nodes)/len(isolated_nodes)
four = sum(connected_components)/len(connected_components)

print("There are " + str(G.number_of_nodes()) + " nodes in this graph.")
print("In average, there are " + str(one) + " edges in this graph.")
print("In average, there are " + str(two) + " triangles in this graph." )
print("In average, there are " + str(three) + " isolated nodes in this graph
print("In average, there are " + str(four) + " connected components in this
```

```
There are 620 nodes in this graph.
In average, there are 2101.0 edges in this graph.
In average, there are 50.8 triangles in this graph.
In average, there are 0.4 isolated nodes in this graph.
In average, there are 1.4 connected components in this graph.
```

Plot the histogram of node degrees for one random graph you generated with bins=15, what's the difference between the node degree sequences of the real graph and the random graph?

In [491…

```
fig, ax = plt.subplots()
ax.hist(degrees,bins=15)
ax.set_title("Histogram of Node Degree")
ax.set_xlabel("Node Degree")

plt.show()


print("One difference is that even though the degrees of the node in the act
```



```
One difference is that even though the degrees of the node in the actual gra
ph is concentrated between 1 and 20, there exist some nodes that have degree
higher than 20 but the random graph has a probability close to 0 of generati
ng such a high degree since the edges are all created randomly with a low pr
obability of 0.01095.
```

A random graph model that fits the degree sequence (10pts)

Given a sequence of expected degrees $(d_1, d_2 \dots d_n)$ of length n, we generate a graph with n nodes, and assigns an edge between node u and node v with probability

$$p_{uv} = \frac{d_u d_v}{\sum_k d_k}$$

.

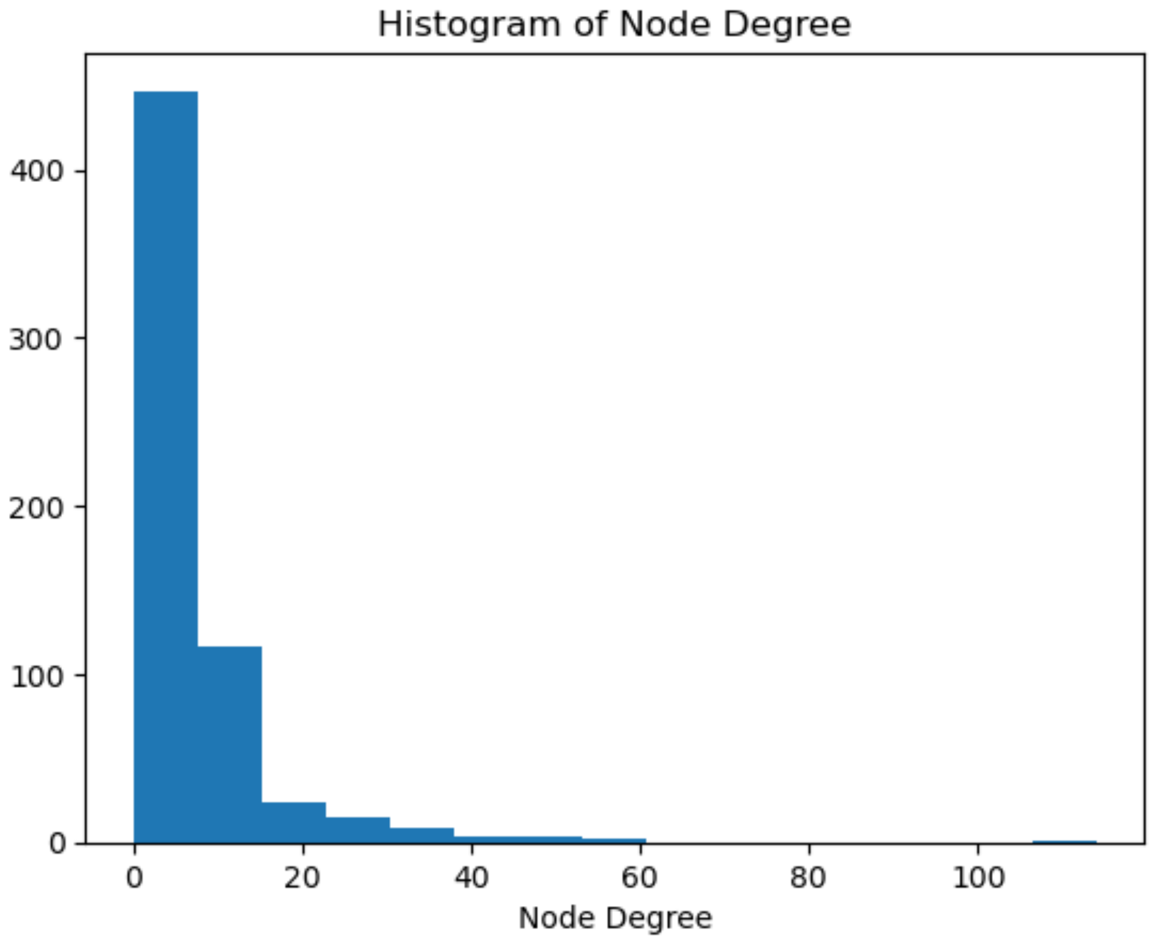
This model is known as the Chung-Lu model and is implemented in the networkx library. To generate a graph from this model, simply use the function `nx.expected_degree_graph(node_degree_list, selfloops=False)`.

To compare the generated graph with a real social network, pass the degree sequence of the real network into this model and generate 10 samples. Then plot the degree histogram of the generated graphs and compare it with the original graph. Additionally, report the same graph statistics for the generated graphs and compare them with the original graph. What observations can we make from this comparison?

In [492…

```
triangles = []
isolated_nodes = []
connected_components = []
edges = []

for x in range(10):
    G = nx.expected_degree_graph(degrees_for_real_data,selfloops=False)
    edges.append(G.number_of_edges())

    tri = nx.triangles(G)
    tri_sum = sum(tri.values())/ 3
    tri_sum = int(tri_sum)
    triangles.append(tri_sum)

    sum_degree = 0
    degrees = [val for (node, val) in G.degree]
    for x in degrees:
        if x == 0:
            sum_degree += 1
    isolated_nodes.append(sum_degree)

    connected_components.append(len(list(nx.connected_components(G))))

one = sum(edges)/len(edges)
two = sum(triangles)/len(triangles)
three = sum(isolated_nodes)/len(isolated_nodes)
four = sum(connected_components)/len(connected_components)

print("There are " + str(G.number_of_nodes()) + " nodes in this graph.")
print("In average, there are " + str(one) + " edges in this graph.")
print("In average, there are " + str(two) + " triangles in this graph." )
print("In average, there are " + str(three) + " isolated nodes in this graph
print("In average, there are " + str(four) + " connected components in this

fig, ax = plt.subplots()
ax.hist(degrees,bins=15)
ax.set_title("Histogram of Node Degree")
ax.set_xlabel("Node Degree")
plt.show()


print("The Chung-Lu model is similar to the actual graph in terms of number
print("However, when looking at statistics, even though the number of edges
```

```
There are 620 nodes in this graph.
In average, there are 2098.0 edges in this graph.
In average, there are 995.6 triangles in this graph.
In average, there are 57.6 isolated nodes in this graph.
In average, there are 59.0 connected components in this graph.
```



```
The Chung-Lu model is similar to the actual graph in terms of number of degr
ees of the nodes when drawing with bins=15. It has a high peak in between 0
to approximately 10 and has a big downfall. There also exist some degrees th
at are high just like the real data.
However, when looking at statistics, even though the number of edges are sim
ilar, the number of triangles, isolated nodes, and connected components are
significantly greater in the Chung-Lu model compared to the real data.
```

In [ ]: