Social Network Programming Assignment 3
By Neelu Verma
(MP19AI002)

February 2, 2022

1 Write Code for the Following:

1. Write functions to Generate Random Graph with (i) N nodes and L edges and (ii) N and p parameter. (Do not use lib function)
2. Generate Random Graph and Scale-Free Graph (using Barabasi-Albert model) of different sizes ranging from N=100 to $10^{5/10}6$ (based on your machine). Plot their degree distributions, both in usual scal and log-log scale.
3. Do a structural analysis of a Random Graph and a Scale-Free Graph of moderate size.

For 2 and 3, you may use the lib functions available with NetworkX

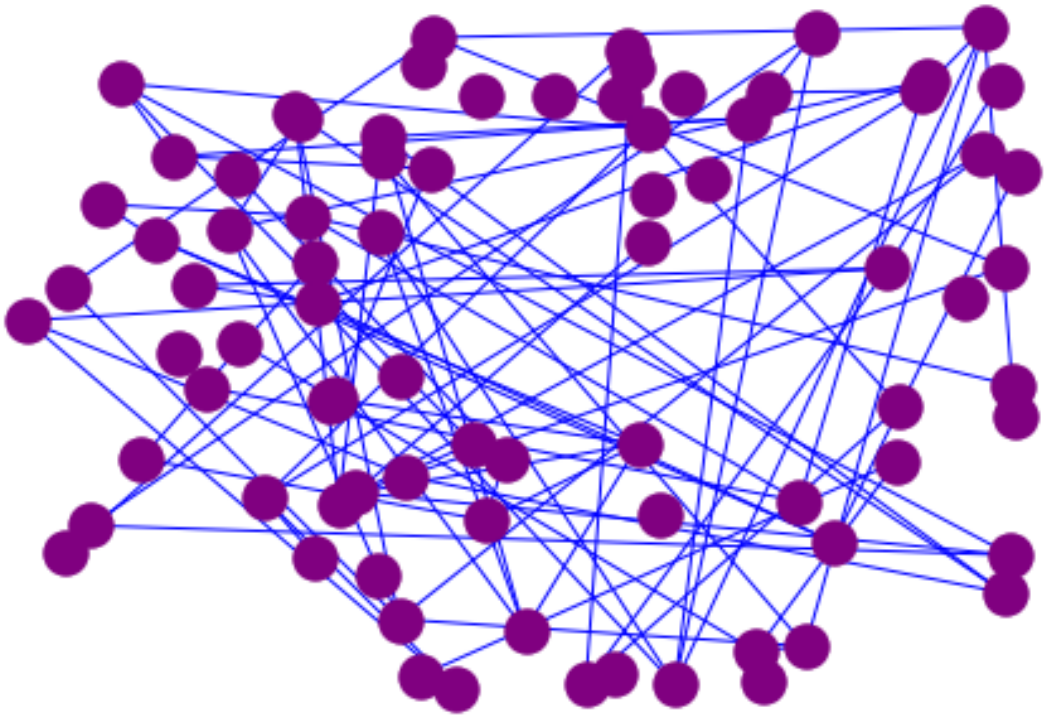
2 N nodes and L edges

[ ]:
```
#importing all related libraries:
import matplotlib.pyplot as plt
import networkx as nx
import random
import collections
import numpy
```

[19]:
```
def Graph_of_Random():
    # here we are taking input from user: In which value of N is number of nodes
    # value of L is a number of edges

    N=int(input("Please Enter the value of N: "))
    L=int(input("Please enter the value of L: "))

    RG = nx.Graph()
    for node in range (0, N):
        RG.add_node(node)

    for i in range(L):
        source=random.randint(1,N)
        target=random.randint(1,N)
```

```
        RG.add_edge(source, target)

    nx.draw(RG, pos=nx.random_layout(RG), node_color='purple', node_shape='o',␣
 ↪edge_color='blue')
    plt.show()

Graph_of_Random()
```

```
Please Enter the value of N: 79
Please enter the value of L: 89
```

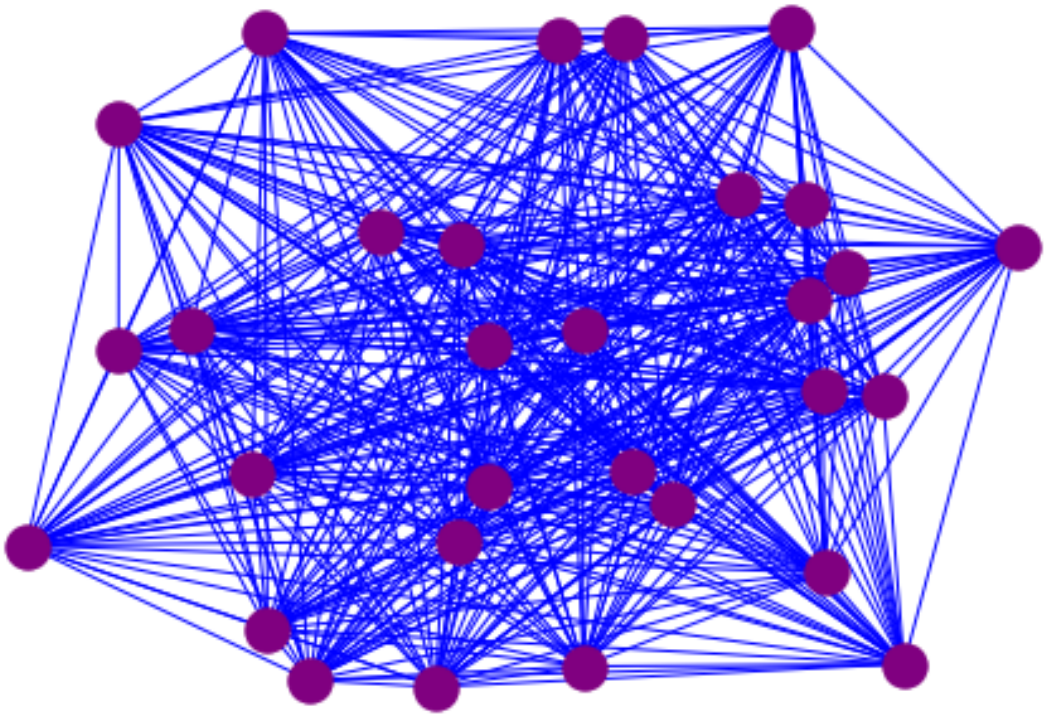


3 N and p parameter.

[29]:
```
def RandomGraph_parameter():
    N=int(input("Please Enter the value of N:"))
    p=int(input("Please Enter the value of p: "))

    PG = nx.Graph()

    for node in range (0, N):
        PG.add_node(node)
```

```
    for node_s in range (0, N):
        for node_t in range  (node_s, N):

            if ((random_number() < p ) and (node_s != node_t)):
                PG.add_edge(node_s, node_t)

    nx.draw(PG, pos=nx.random_layout(PG), node_color='purple', node_shape='o',␣
↪edge_color='blue')
    plt.show()

RandomGraph_parameter()
```

```
Please Enter the value of N:30
Please Enter the value of p: 10
```

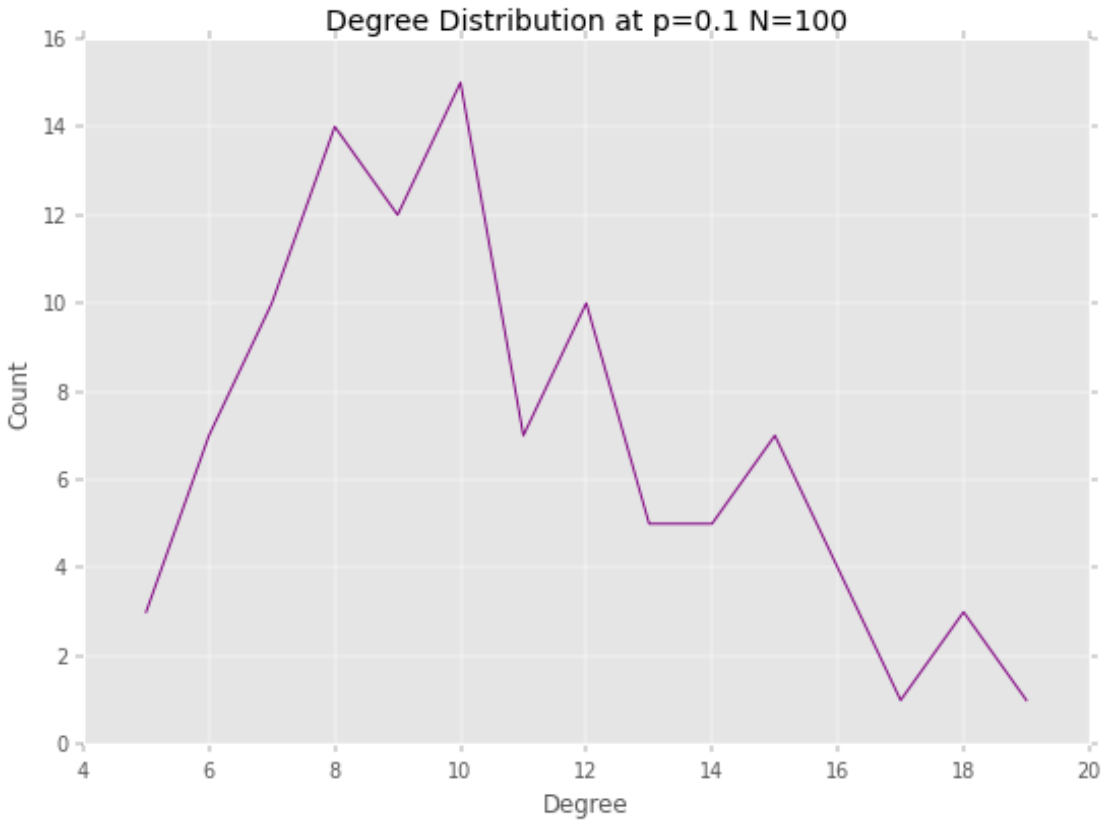


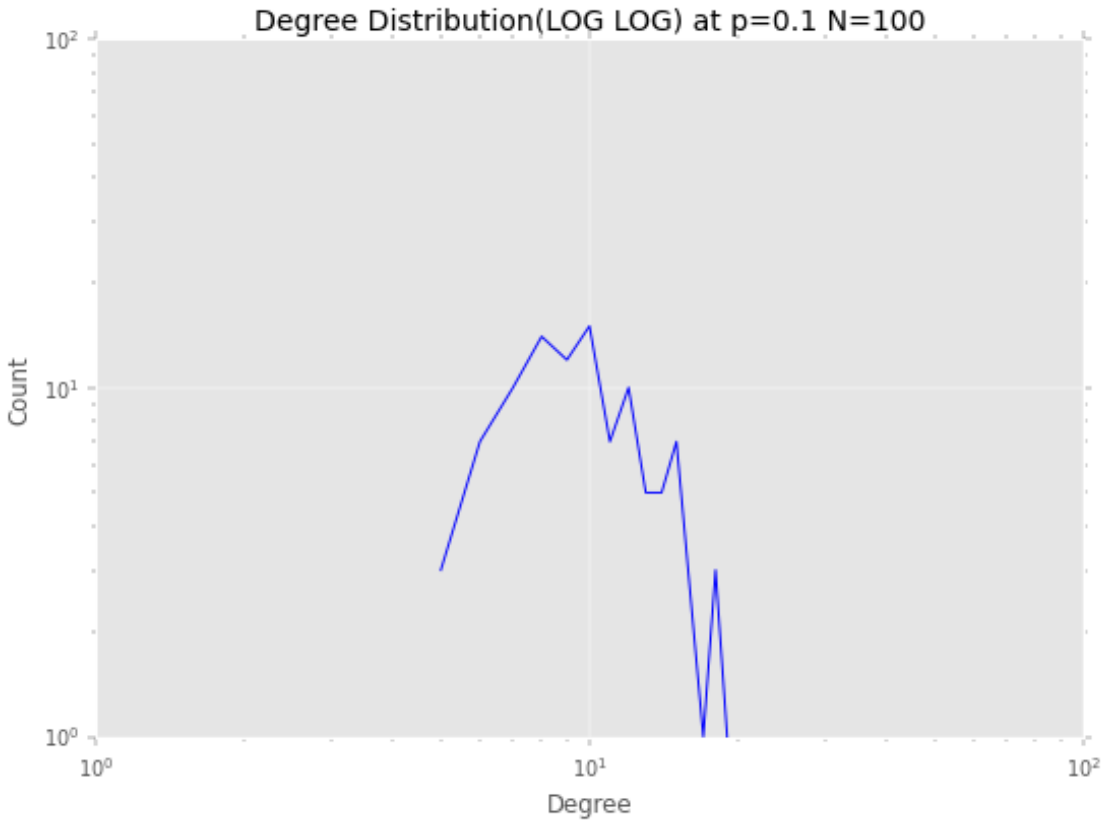
4 Q2. Generate Random Graph and Scale-Free Graph (using Barabasi-Albert model) of different sizes ranging from N=100 to $10^{5/10}6$.

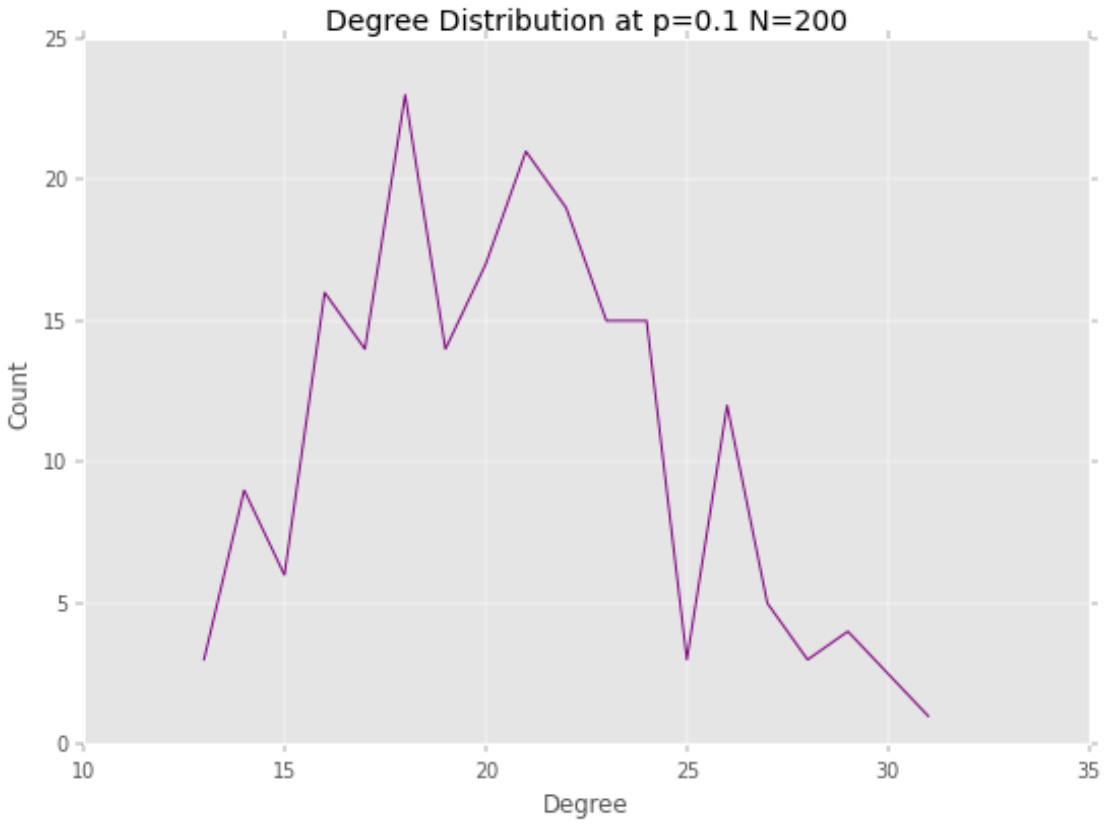
Plot their degree distributions, both in usual scal and log-log scale.

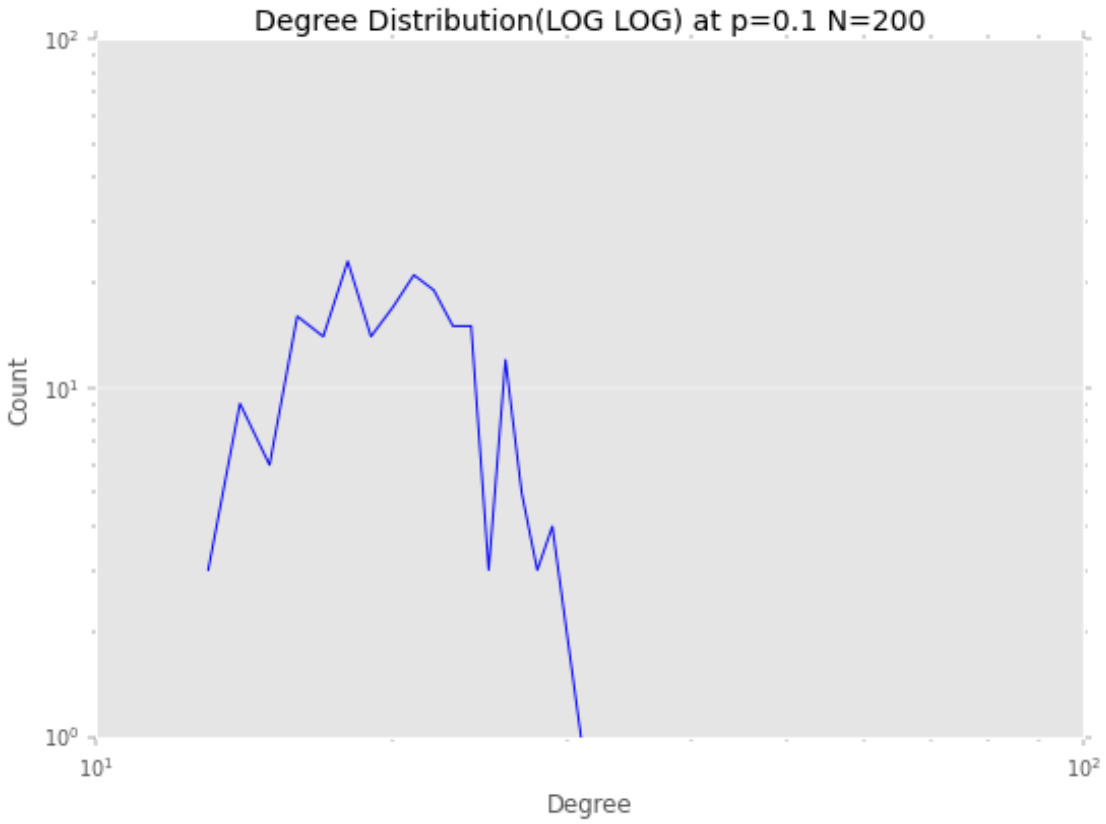
[76]:
```
#function for Generate Random Graph
```

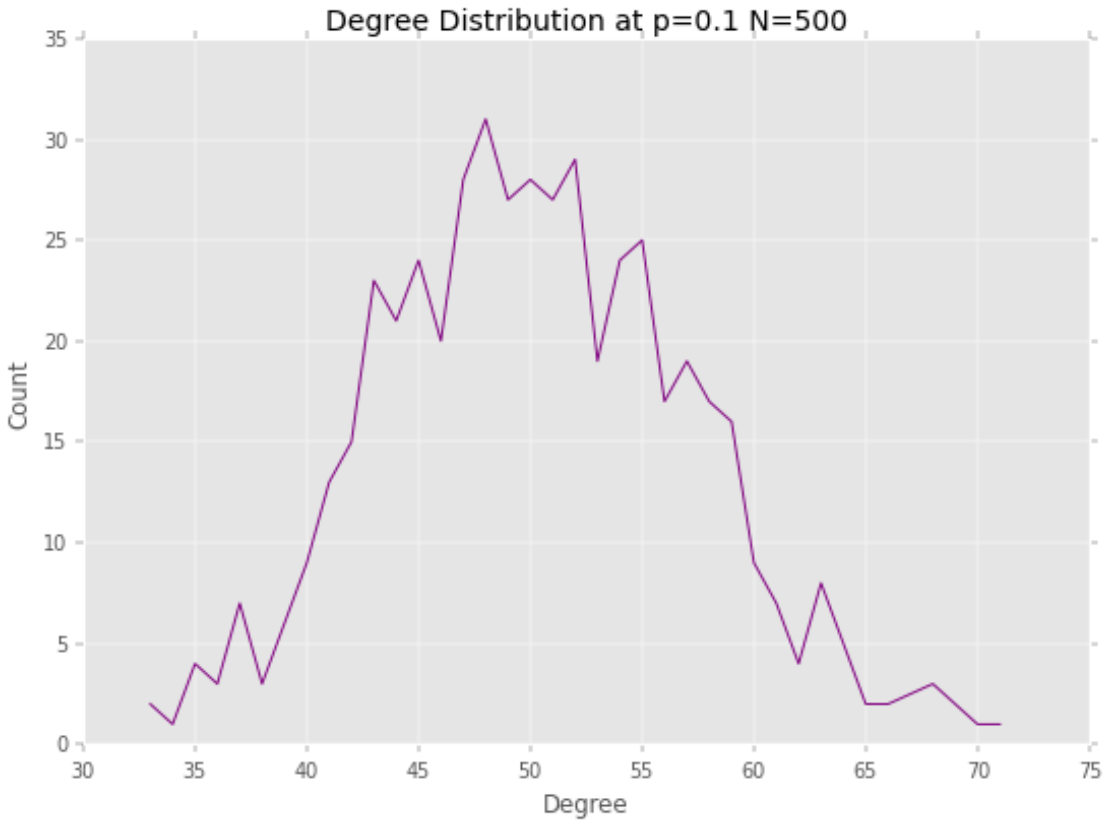
```
Array_of_Nodes=[100,200,500,1000,2000,5000,10000]

for N in Array_of_Nodes:
    Generate_RN=nx.gnp_random_graph(N, 0.1, seed=None, directed=False)


    deg_seq = sorted([d for n, d in Generate_RN.degree()], reverse=True)
    DC = collections.Counter(deg_seq)
    deg, cnt = zip(*DC.items())

 ########################Plot for Degree Distribution on usual scale at p=0.
 ↪1########
    fig, ax = plt.subplots(dpi=70)
    plt.plot(deg, cnt,'purple')
    plt.title("Degree Distribution at p=0.1 N="+str(N))
    plt.ylabel("Count")
    plt.xlabel("Degree")
    plt.tight_layout()
#     plt.xlim(1,180)
    plt.grid("off")

########################Plot for Degree Distribution on LOG LOG at p=0.
 ↪1##############
    fig, ax = plt.subplots(dpi=70)
    plt.loglog(deg, cnt,'blue')
    plt.title("Degree Distribution(LOG LOG) at p=0.1 N="+str(N))
    plt.ylabel("Count")
    plt.xlabel("Degree")
    plt.tight_layout()
#     plt.xlim(1,180)
    plt.grid("off")
```

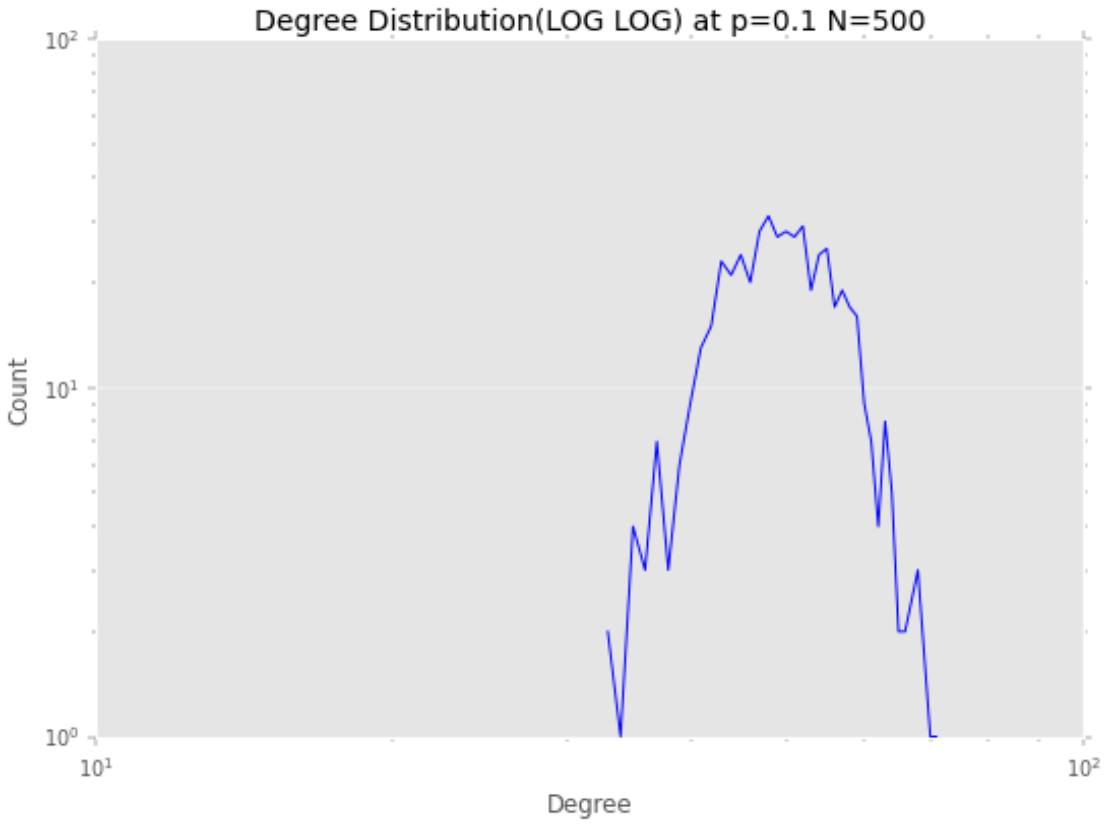

Degree Distribution at p=0.1 N=100
Count
Degree
0
2
4
6
8
10
12
14
16
4
6
8
10
12
14
16
18
20

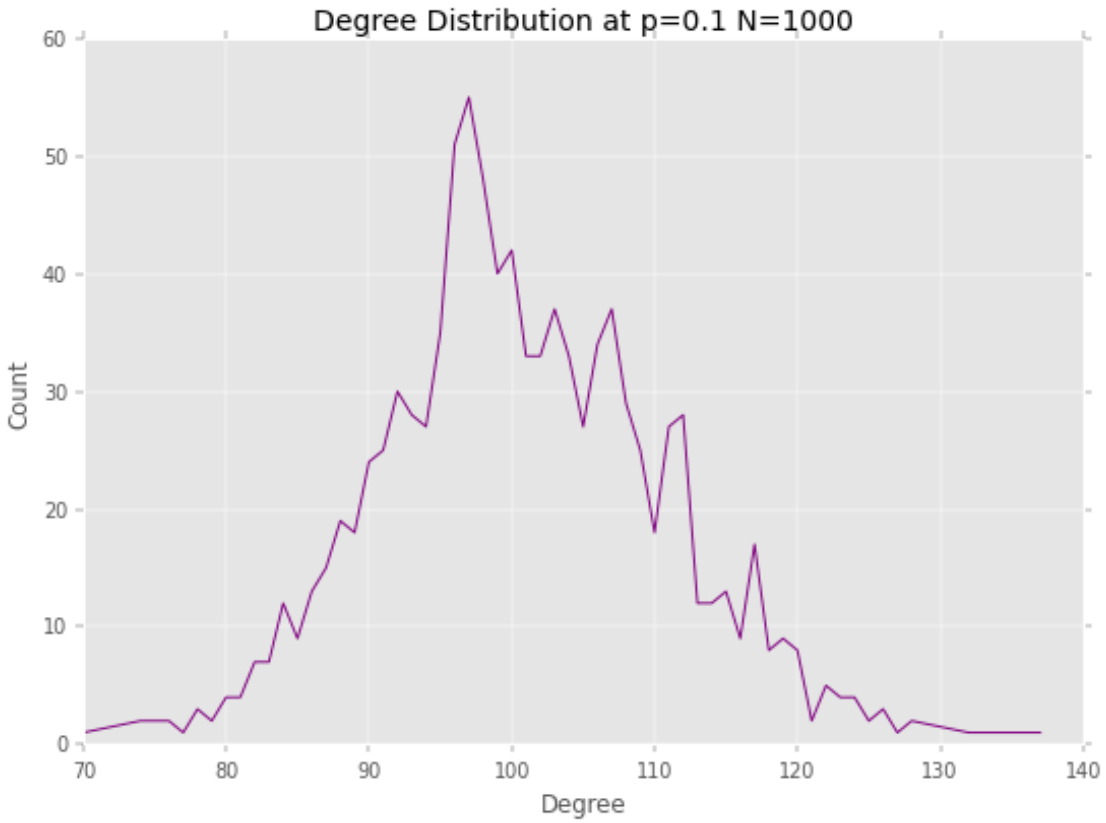

Degree Distribution(LOG LOG) at p=0.1 N=100
Count
Degree
10^0
10^1
10^2
10^0
10^1
10^2

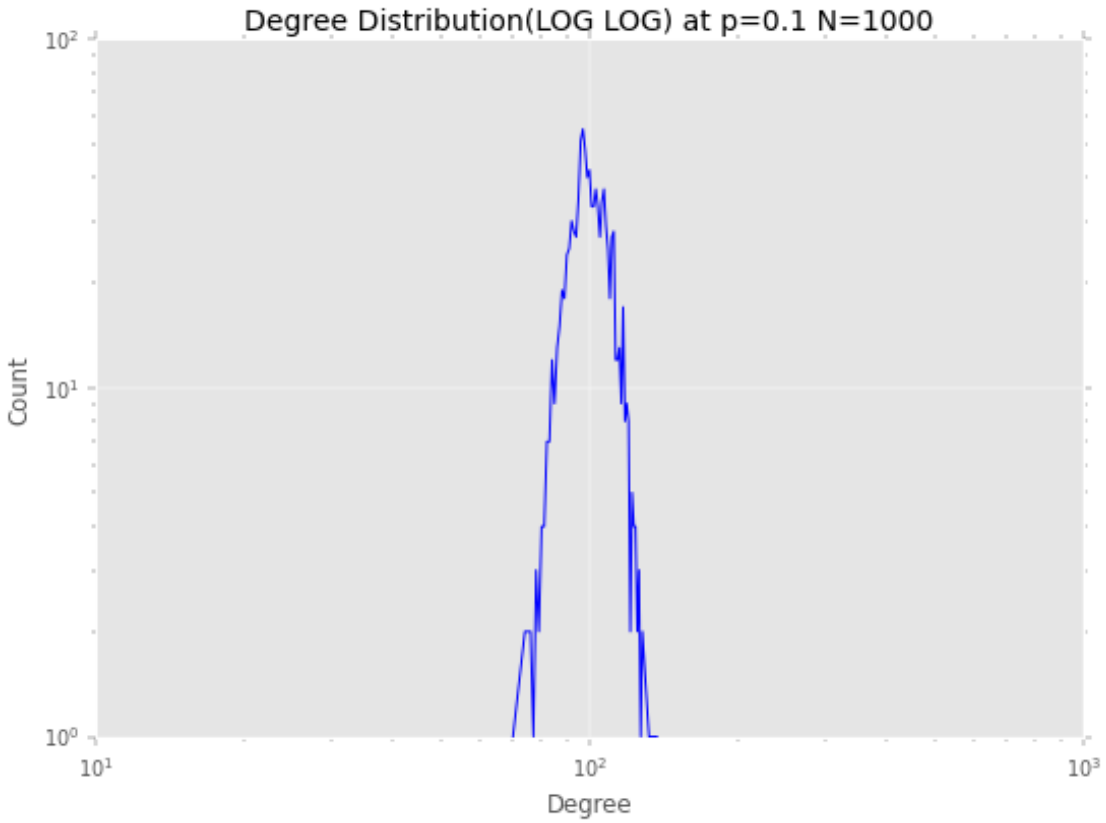

Degree Distribution at p=0.1 N=200
Count
Degree
0
5
10
15
20
25
10
15
20
25
30
35

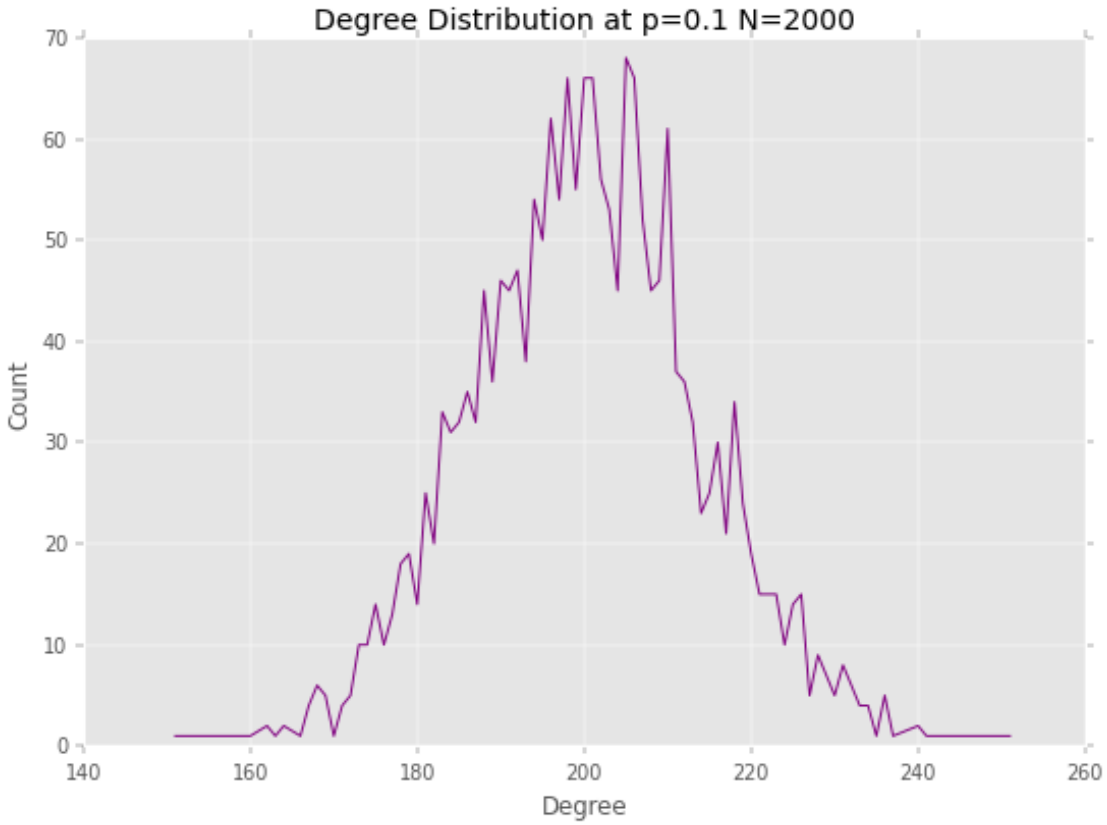

Degree Distribution(LOG LOG) at p=0.1 N=200
Count
Degree
10^2
10^1
10^0
10^1
10^2

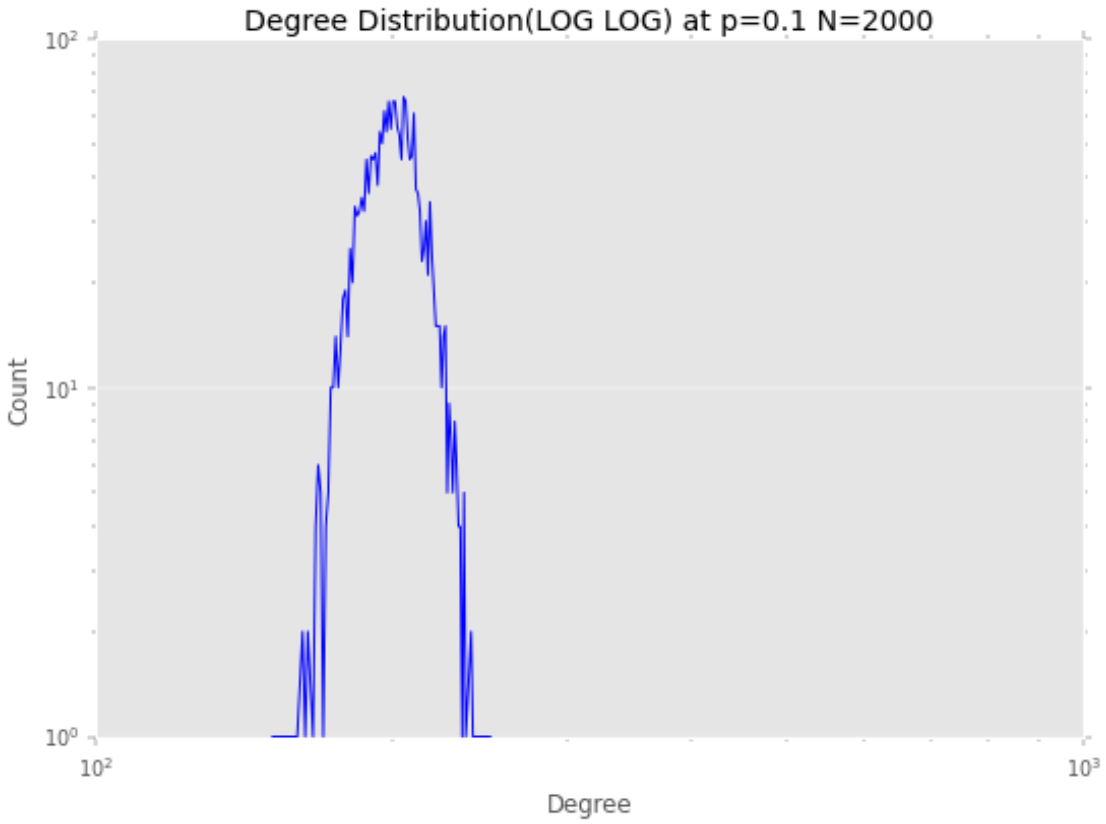

Degree Distribution at p=0.1 N=500
Count
Degree
0
5
10
15
20
25
30
35
30
35
40
45
50
55
60
65
70
75

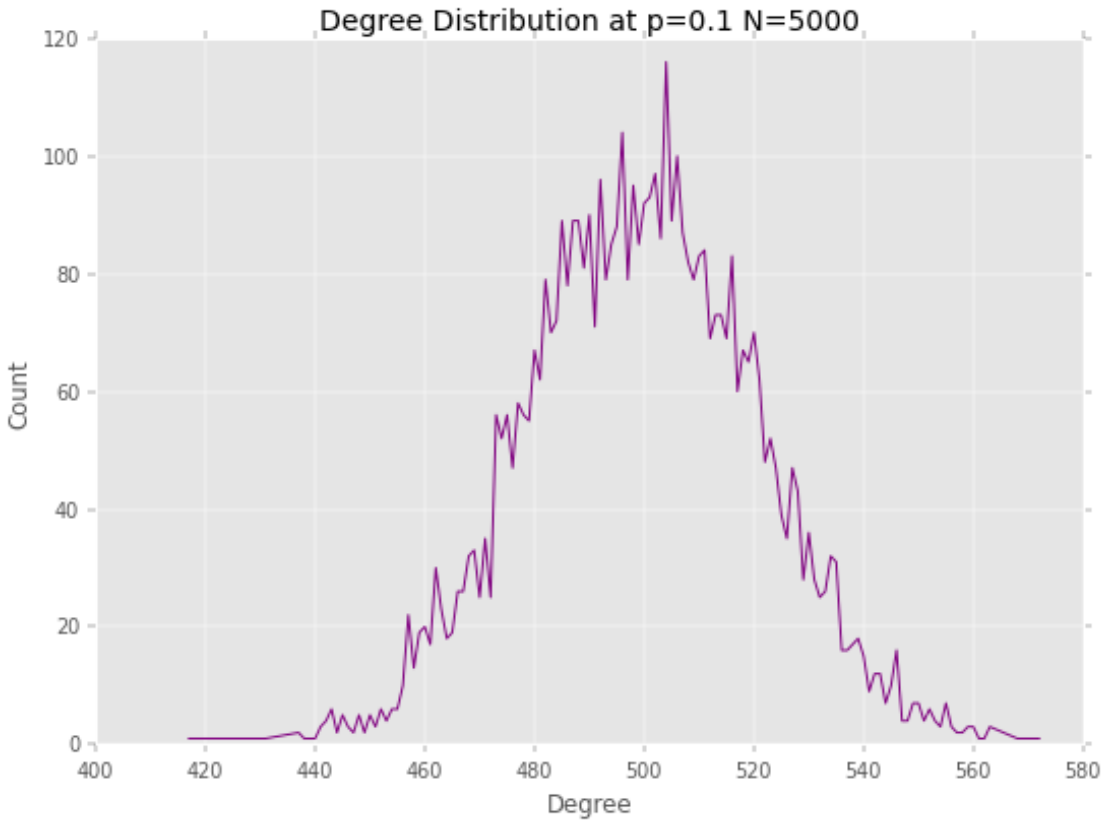

Degree Distribution(LOG LOG) at p=0.1 N=500
Count
Degree
10^2
10^1
10^0
10^1
10^2

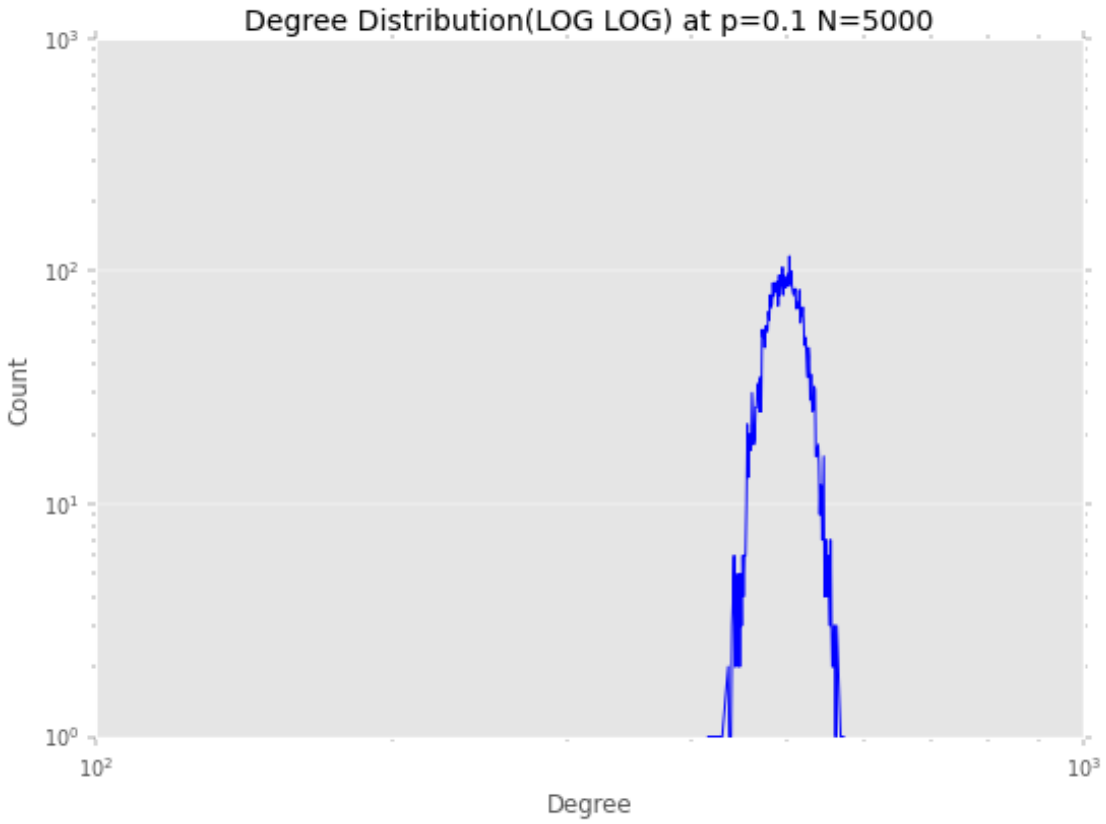

Degree Distribution at p=0.1 N=1000
Count
Degree
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140

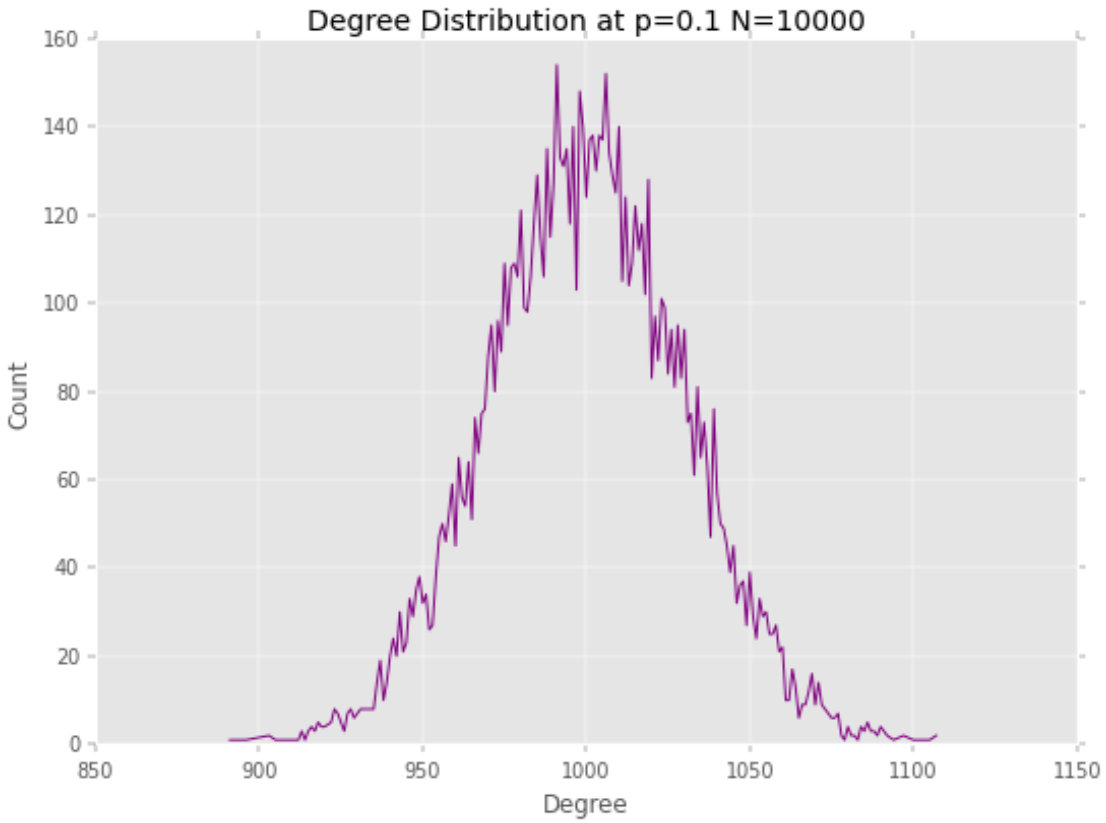

Degree Distribution(LOG LOG) at p=0.1 N=1000
Count
Degree
10^0
10^1
10^2
10^1
10^2
10^3

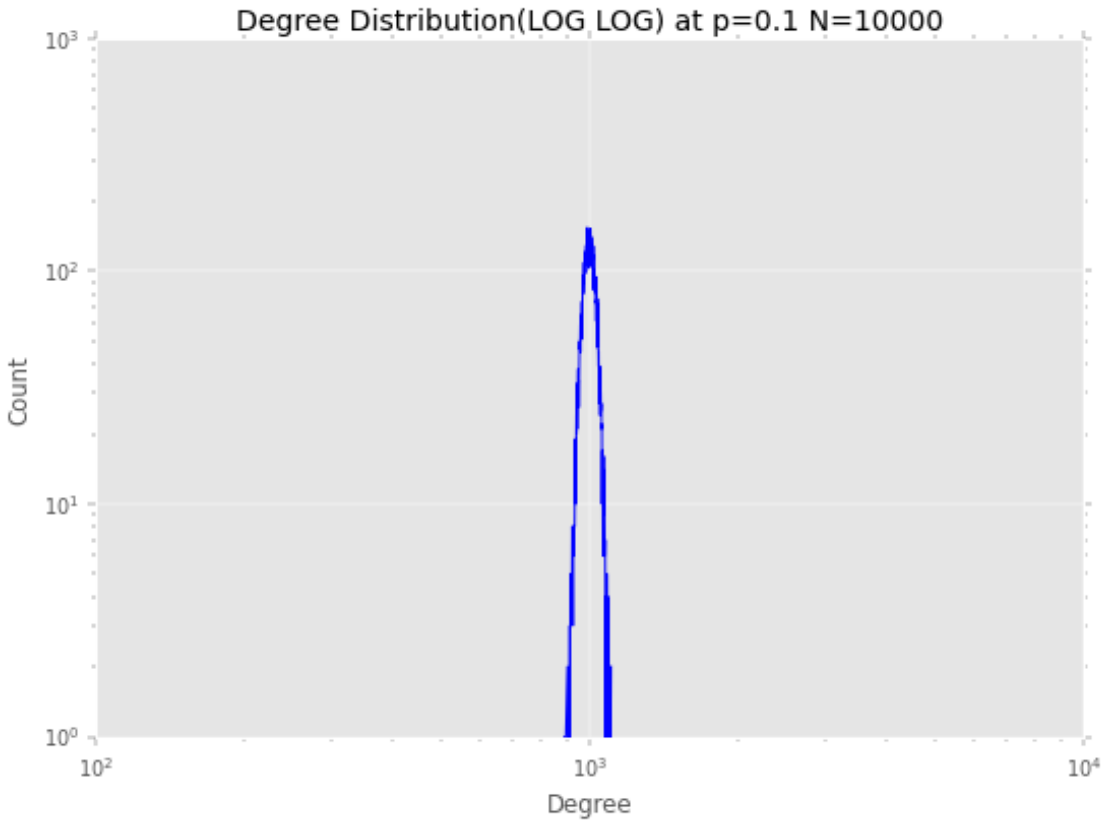

Degree Distribution at p=0.1 N=2000
Count
Degree
0
10
20
30
40
50
60
70
140
160
180
200
220
240
260


Degree Distribution(LOG LOG) at p=0.1 N=2000
Count
Degree
10^2
10^1
10^0
10^2
10^3


Degree Distribution at p=0.1 N=5000
Count
Degree
0
20
40
60
80
100
120
400
420
440
460
480
500
520
540
560
580

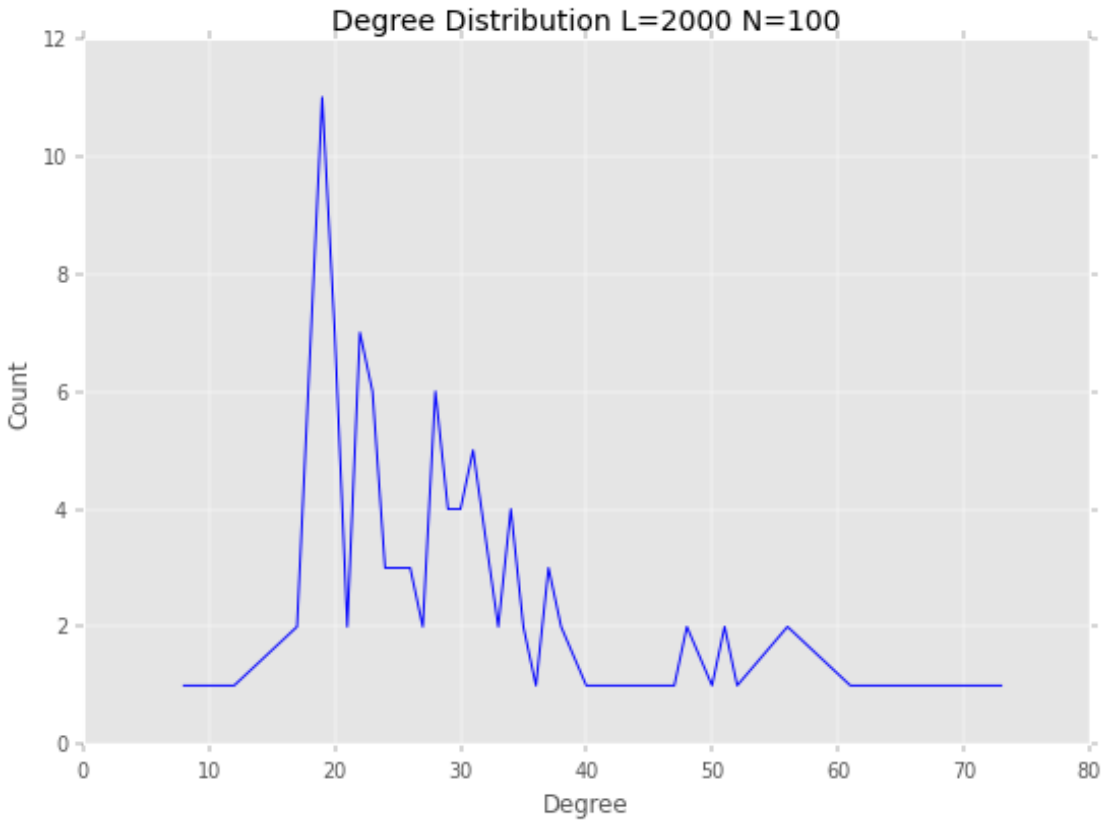

Degree Distribution(LOG LOG) at p=0.1 N=5000
Count
Degree
10^3
10^2
10^1
10^0
10^2
10^3

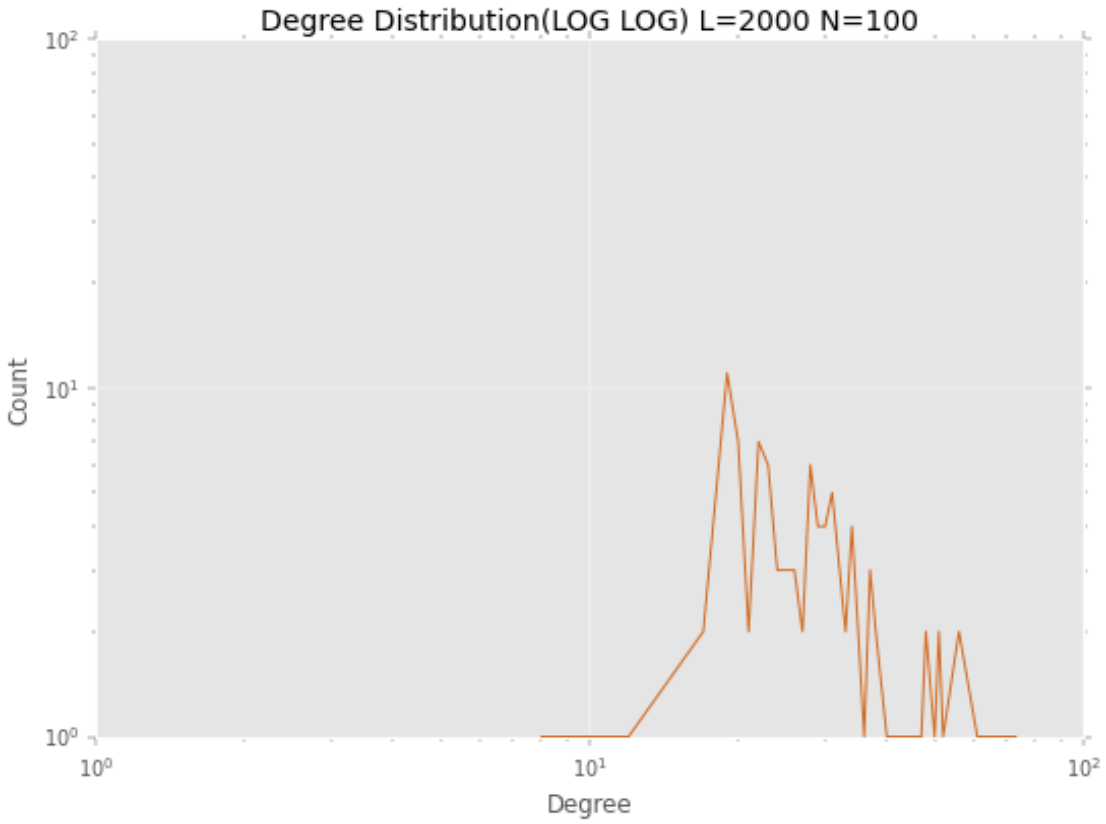

Degree Distribution at p=0.1 N=10000
Count
Degree
0
20
40
60
80
100
120
140
160
850
900
950
1000
1050
1100
1150

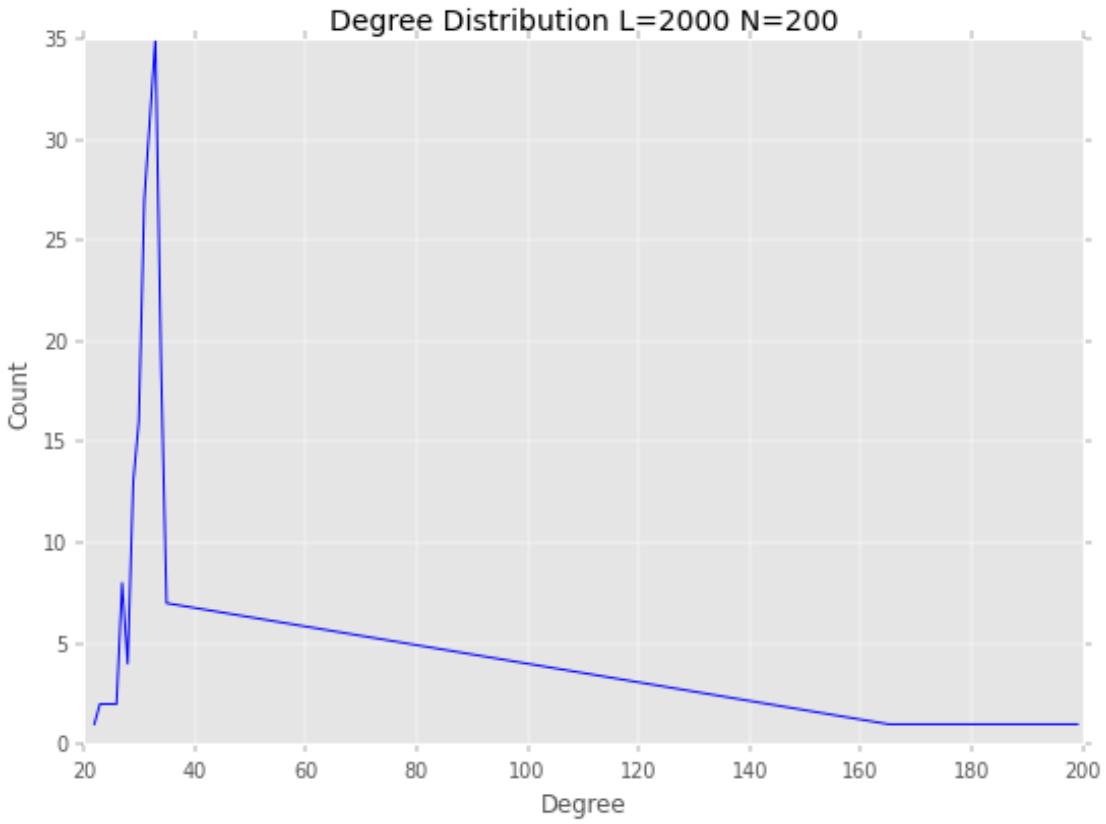


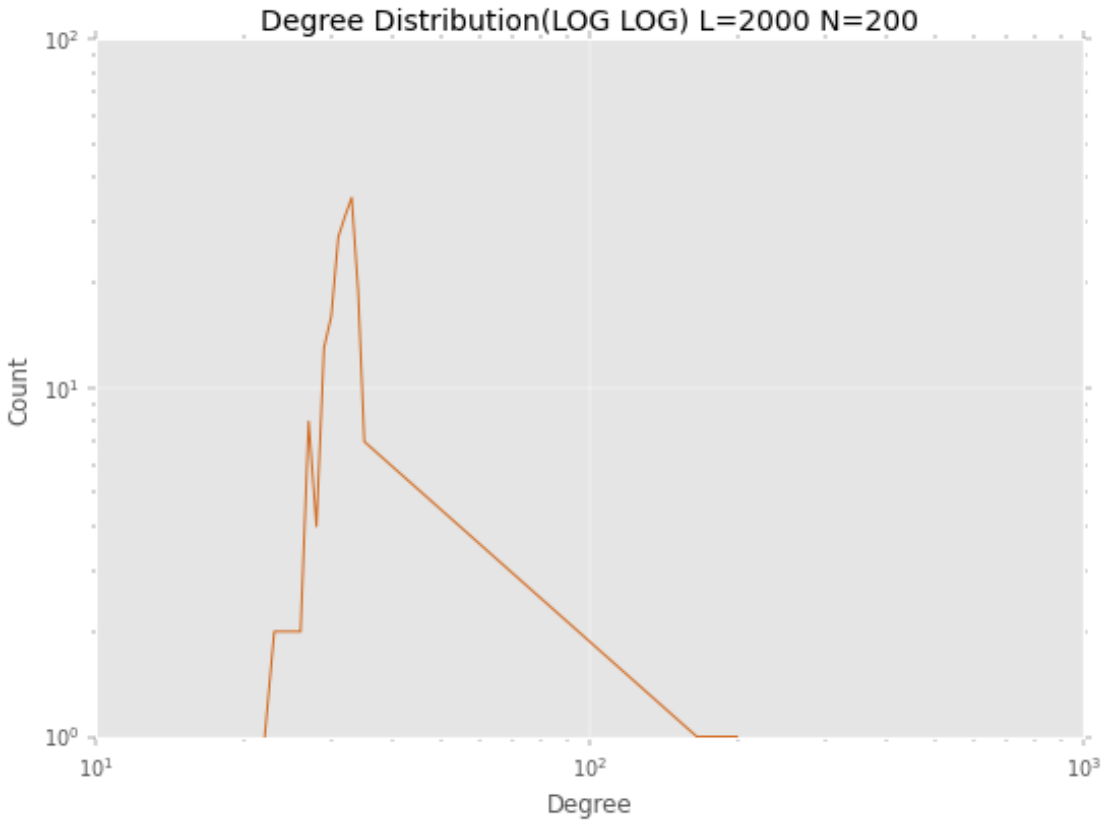
[77]:
```
#function for Scale-Free Graph(using Barabasi-Albert model)

for N in Array_of_Nodes:

    L = random.randint(1,  N-1)

    Generate_BA =  nx.barabasi_albert_graph( N,  L, seed=None)

    deg_seq = sorted([d for n, d in Generate_BA.degree()], reverse=True)
    DC = collections.Counter(deg_seq)
    deg, cnt = zip(*DC.items())


    ␣
 ↪##############################################################################################
    fig, ax = plt.subplots(dpi=70)
    plt.plot(deg, cnt, 'b')
    plt.title("Degree Distribution L=2000 N="+str(N))
    plt.ylabel("Count")
    plt.xlabel("Degree")
    plt.tight_layout()
#     plt.xlim(1,180)
```

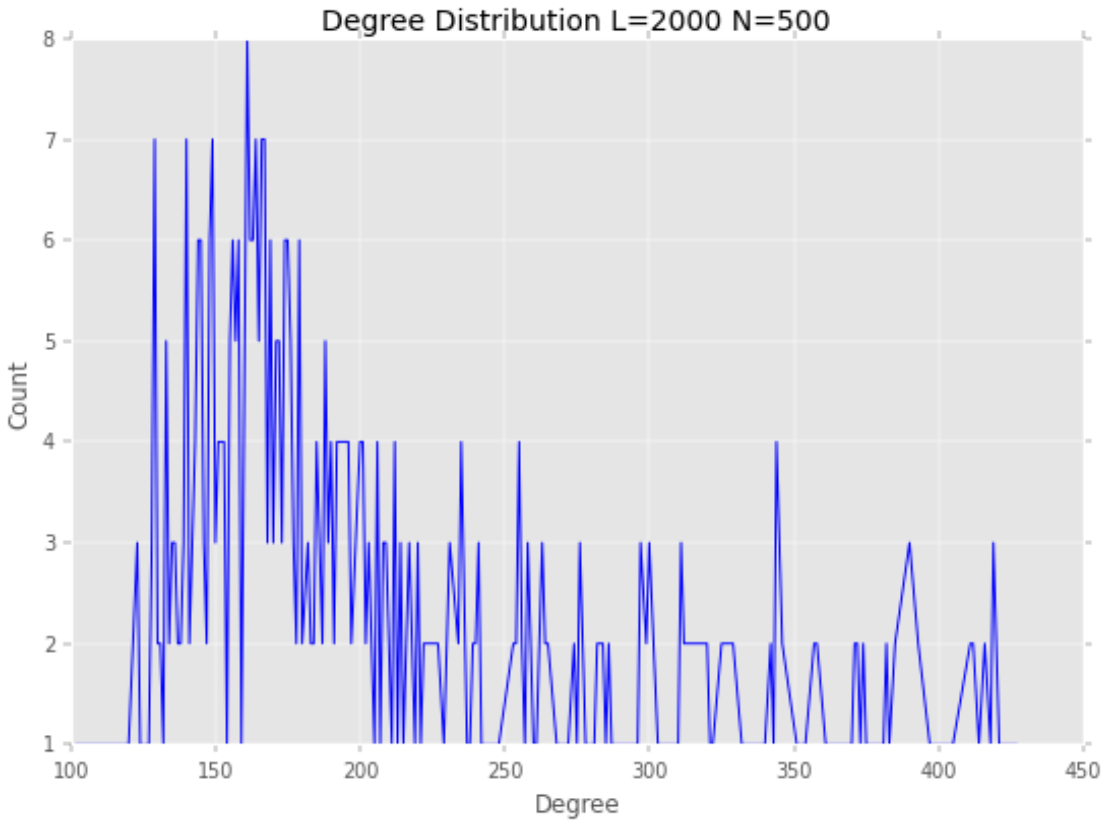
```
    plt.grid("off")


    
##############################################################################################
    fig, ax = plt.subplots(dpi=70)
    plt.loglog(deg, cnt, 'Chocolate')
    plt.title("Degree Distribution(LOG LOG) L=2000 N="+str(N))
    plt.ylabel("Count")
    plt.xlabel("Degree")
    plt.tight_layout()
#     plt.xlim(1,180)
    plt.grid("off")
```

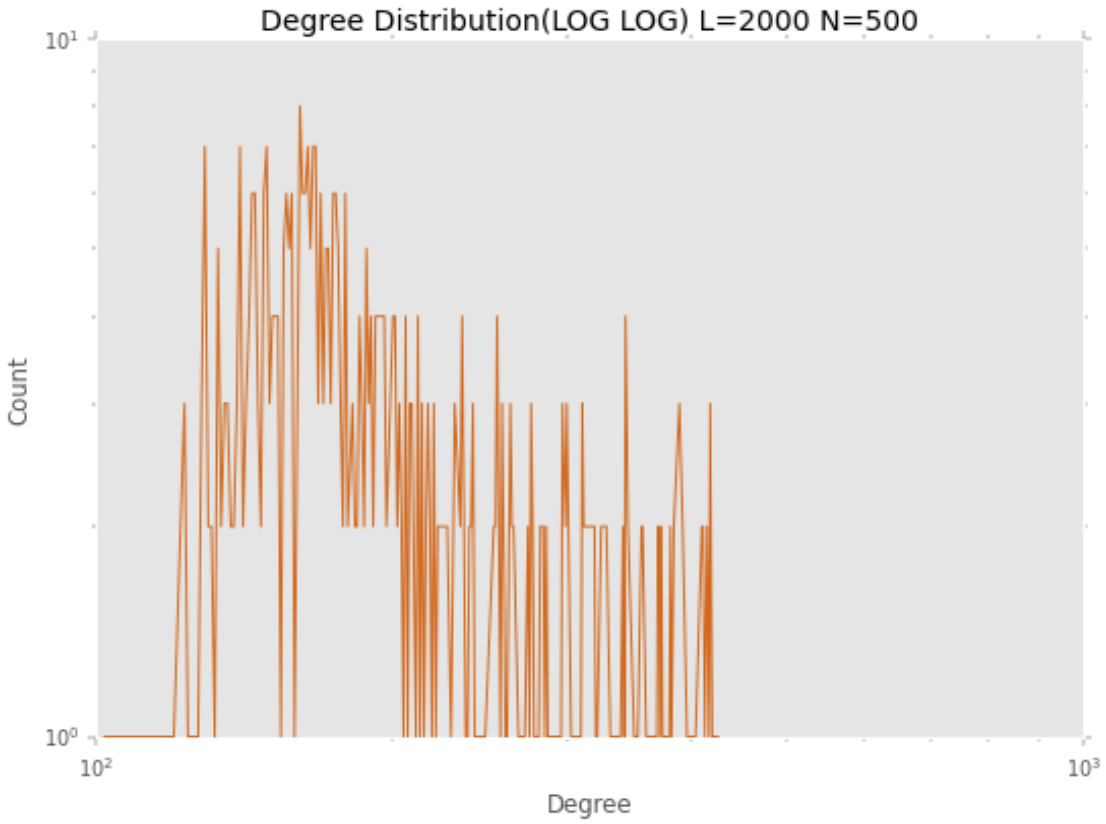


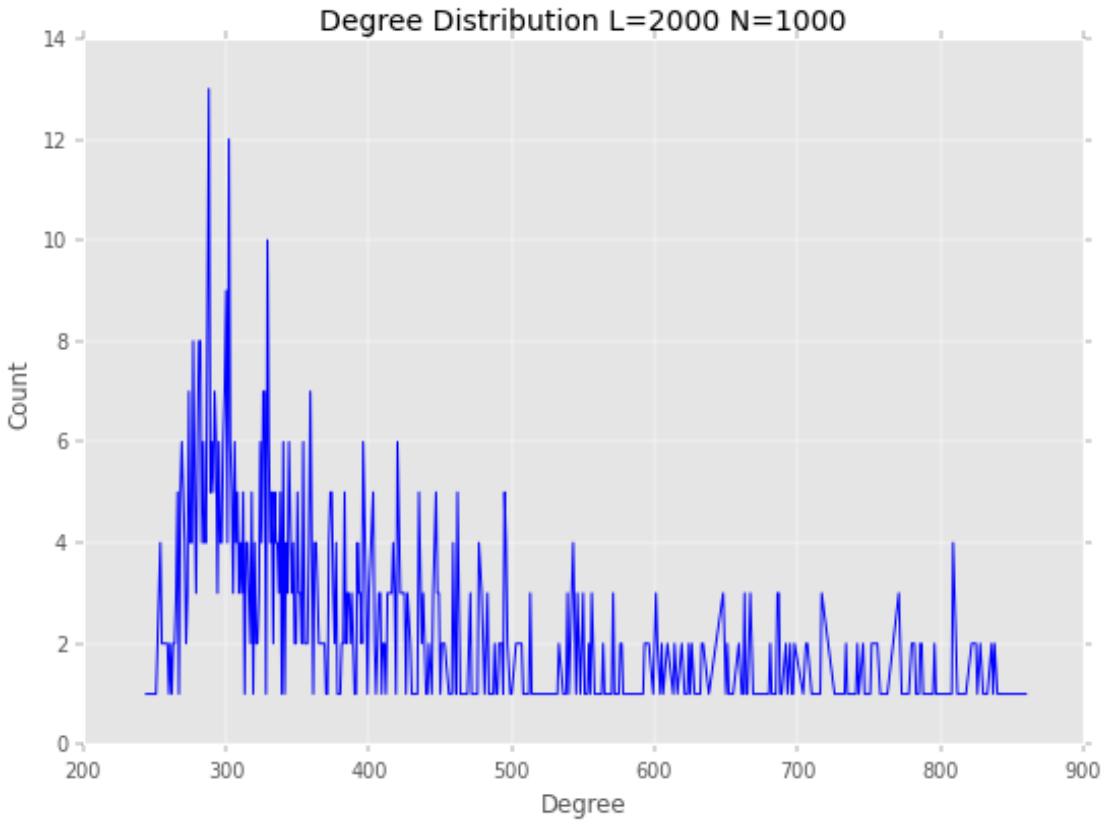

Degree Distribution(LOG LOG) L=2000 N=100
Count
Degree
10^0
10^1
10^2

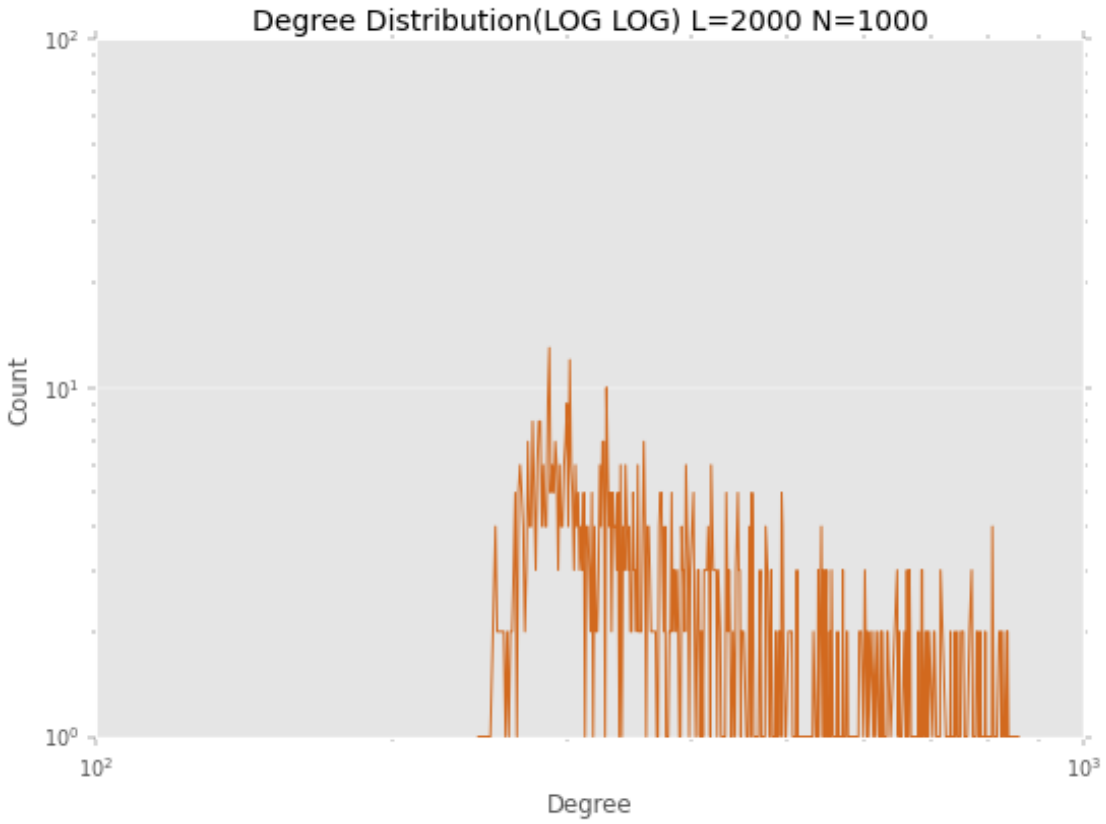

Degree Distribution L=2000 N=200
Count
Degree
0
5
10
15
20
25
30
35
20
40
60
80
100
120
140
160
180
200

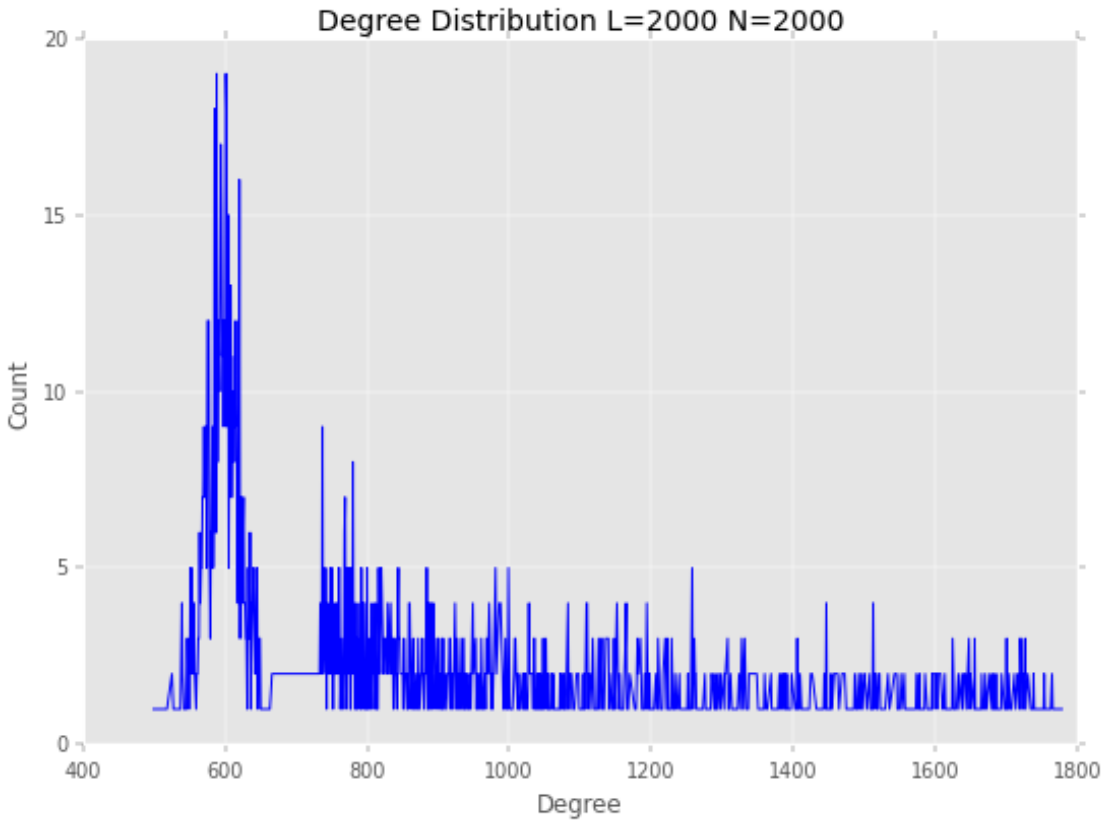

Degree Distribution(LOG LOG) L=2000 N=200
Count
Degree
10^0
10^1
10^2
10^1
10^2
10^3

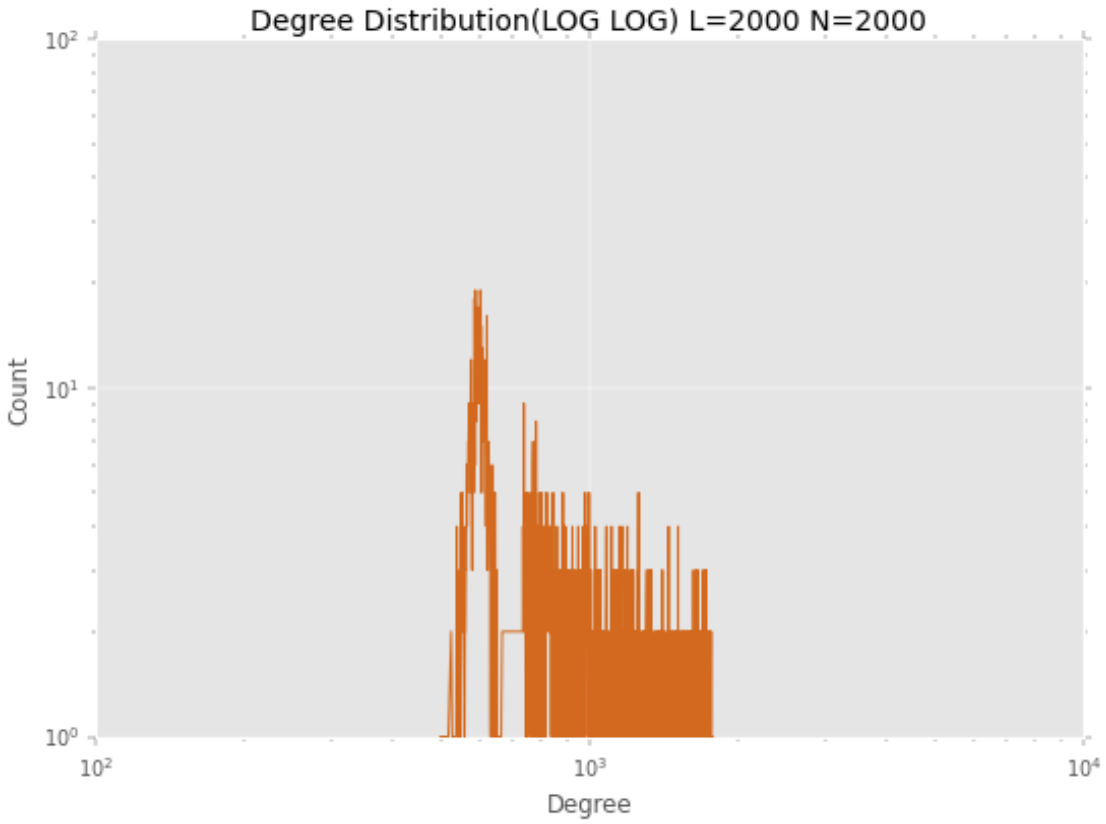

Degree Distribution L=2000 N=500
Count
Degree
1
2
3
4
5
6
7
8
100
150
200
250
300
350
400
450

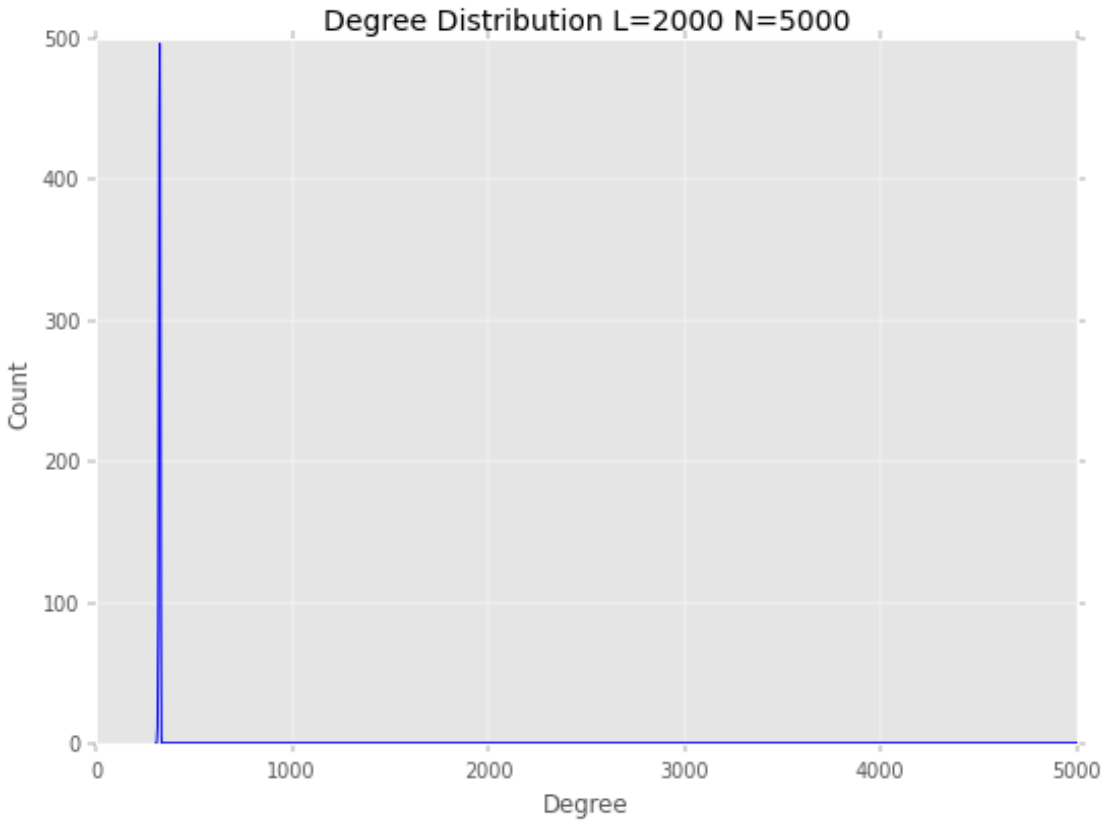

Degree Distribution(LOG LOG) L=2000 N=500
Count
Degree
10^1
10^0
10^2
10^3

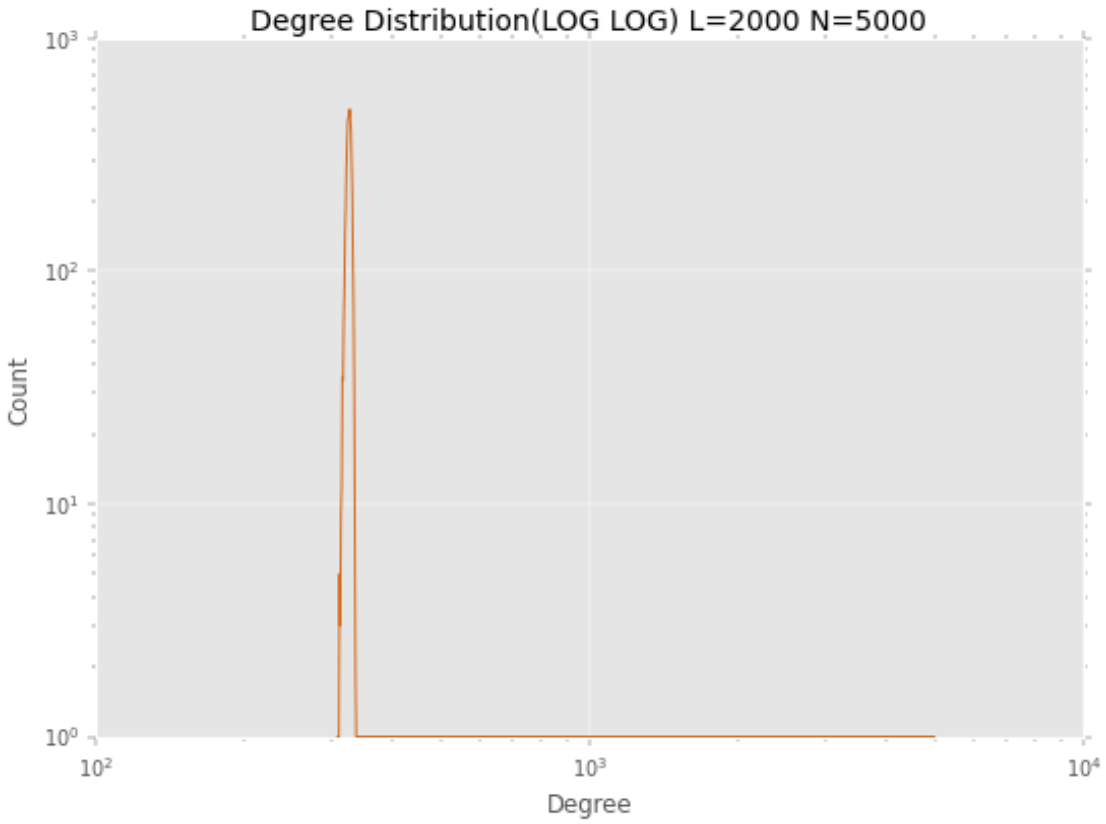

Degree Distribution L=2000 N=1000
Count
Degree
0
2
4
6
8
10
12
14
200
300
400
500
600
700
800
900

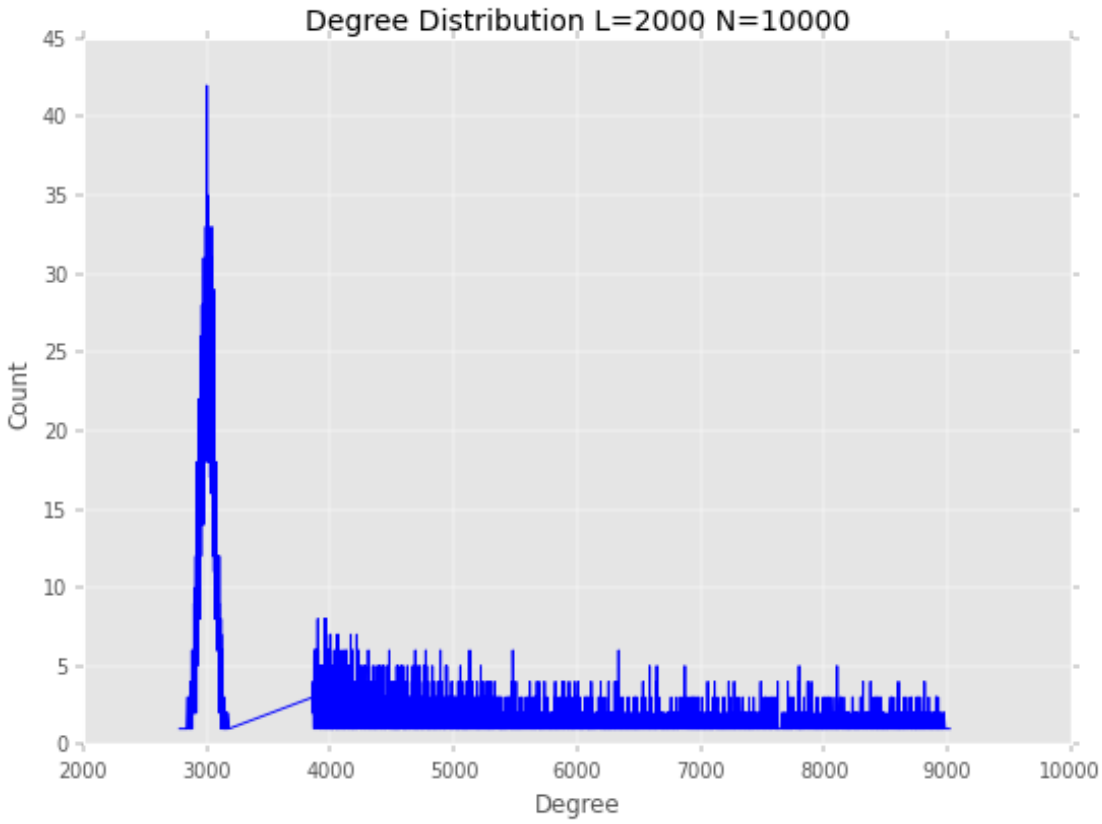

Degree Distribution(LOG LOG) L=2000 N=1000
Count
Degree
10^2
10^1
10^0
10^2
10^3

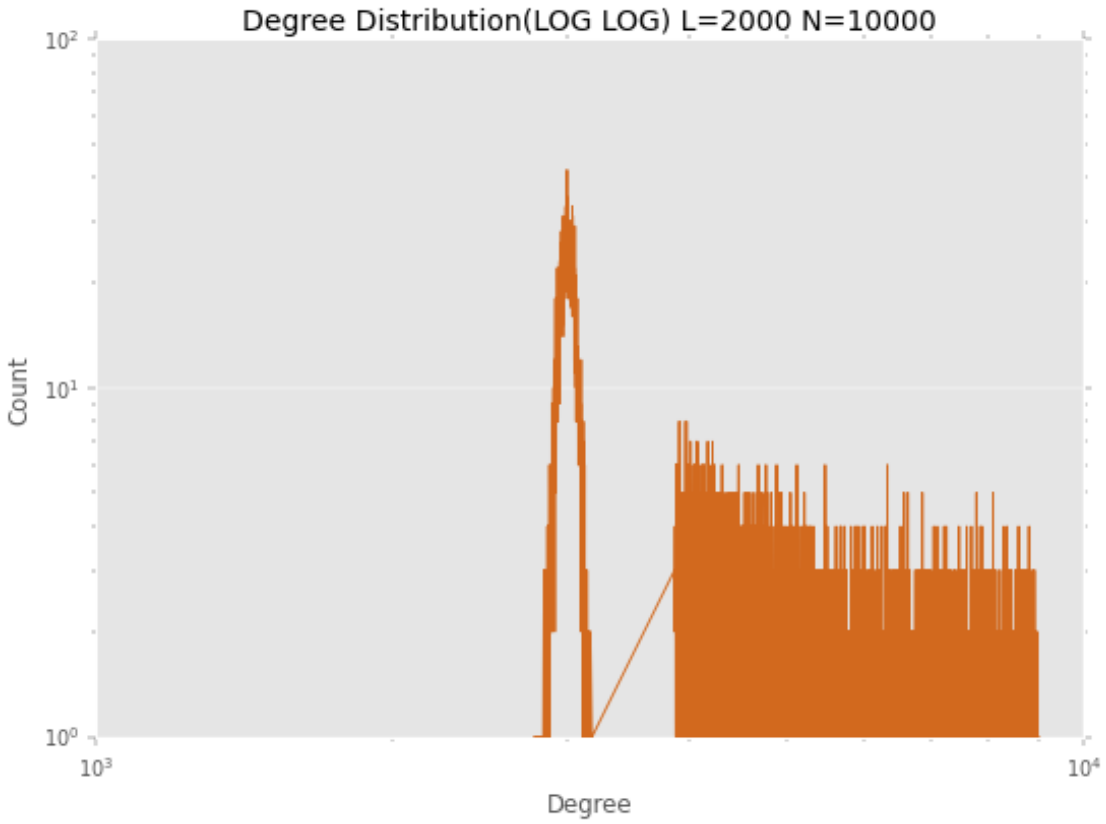

Degree Distribution L=2000 N=2000
Count
Degree
0
5
10
15
20
400
600
800
1000
1200
1400
1600
1800


Degree Distribution(LOG LOG) L=2000 N=2000
Count
Degree
10^0
10^1
10^2
10^3
10^4


Degree Distribution L=2000 N=5000
Count
Degree
0
100
200
300
400
500
0
1000
2000
3000
4000
5000


Degree Distribution(LOG LOG) L=2000 N=5000
Count
Degree
10^0
10^1
10^2
10^3
10^2
10^3
10^4


Degree Distribution L=2000 N=10000
Count
Degree
0
5
10
15
20
25
30
35
40
45
2000
3000
4000
5000
6000
7000
8000
9000
10000

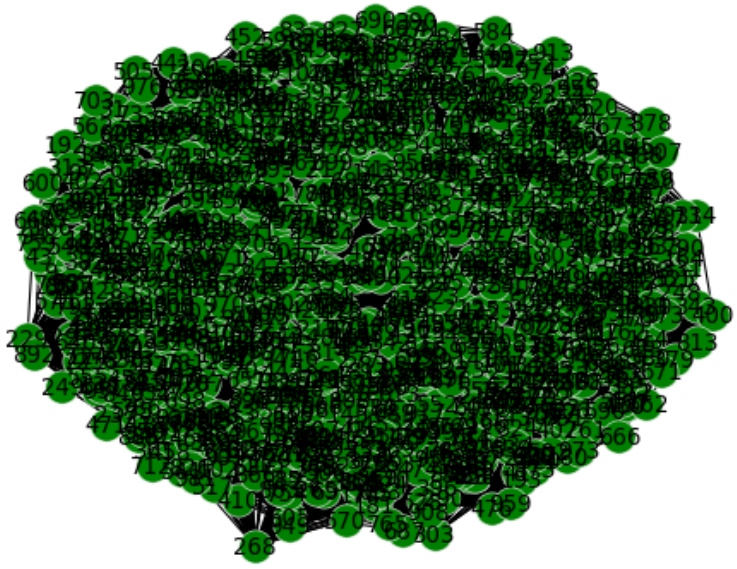


5 Q3. Structural analysis of a Random Graph .

[78]:
```
#structural analysis of a Random Graph

Generated_Random_Network=nx.gnp_random_graph(1000, 0.1, seed=None,␣
 ↪directed=False)
print(nx.info(Generated_Random_Network))

nx.draw(Generated_Random_Network, with_labels = True, node_color ='green' )

print("The Diameter for random graph is: "+str(nx.
 ↪diameter(Generated_Random_Network)))

t1 = nx.triangles(Generated_Random_Network)
sumt1=sum(t1.values())
triangle1=sumt1/3

print("Number of triangles for random graph is: "+str(triangle1))
```

```
print("Number of connected components for random graph is: "+str(nx.
 ↪number_connected_components(Generated_Random_Network)))
print("Clustering coefficient for random graph is: "+str(nx.
 ↪average_clustering(Generated_Random_Network)))

degree_sequence1 = sorted([d for n, d in Generated_Random_Network.degree()],␣
 ↪reverse=True)  # degree sequence
degreeCount1 = collections.Counter(degree_sequence1)
deg1, cnt1 = zip(*degreeCount1.items())

deg1=list(deg1)
print("Average degree for random graph is: "+str(numpy.average(deg1)))
```

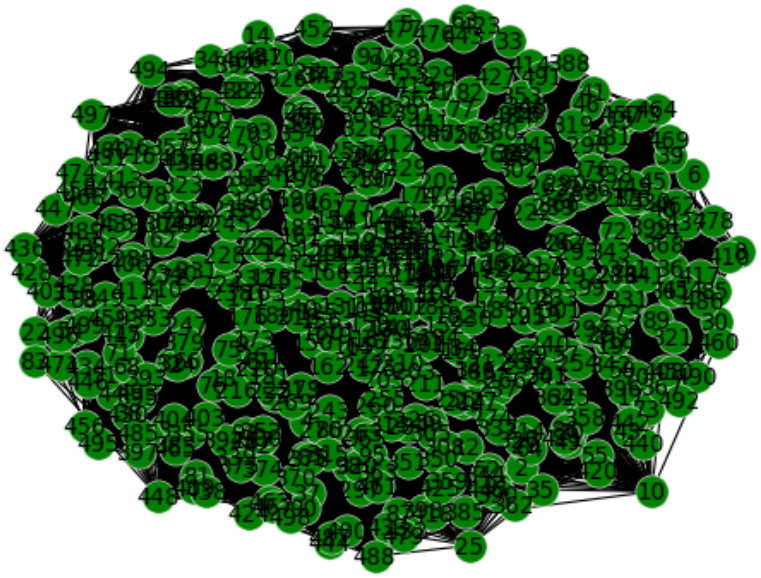
```
Name:
Type: Graph
Number of nodes: 1000
Number of edges: 49958
Average degree:  99.9160
The Diameter for random graph is: 3
Number of triangles for random graph is: 166164.0
Number of connected components for random graph is: 1
Clustering coefficient for random graph is: 0.10002015768402073
Average degree for random graph is: 101.26315789473684
```



6 Q3. Structural analysis of a Scale-Free Graph of moderate size

[79]:
```
Generated_Scale_free=nx.barabasi_albert_graph(500, 100, seed=None)

print(nx.info(Generated_Scale_free))

nx.draw(Generated_Scale_free, with_labels = True, node_color ='green' )

print("The Diameter for scale free graph is: "+str(nx.
 ↪diameter(Generated_Scale_free)))

t2 = nx.triangles(Generated_Scale_free)
sumt2=sum(t2.values())
triangle2=sumt2/3

print("Number of triangles for scale free graph is: "+str(triangle2))
print("Number of connected components for scale free graph is: "+str(nx.
 ↪number_connected_components(Generated_Scale_free)))
print("Clustering coefficient for scale free graph is: "+str(nx.
 ↪average_clustering(Generated_Scale_free)))


degree_sequence2 = sorted([d for n, d in Generated_Scale_free.degree()],␣
 ↪reverse=True)

degreeCount2 = collections.Counter(degree_sequence2)
deg2, cnt2 = zip(*degreeCount2.items())
deg2=list(deg2)

print("Average degree for scale free graph is: "+str(numpy.average(deg2)))
```

```
Name:
Type: Graph
Number of nodes: 500
Number of edges: 40000
Average degree: 160.0000
The Diameter for scale free graph is: 2
Number of triangles for scale free graph is: 996288.0
Number of connected components for scale free graph is: 1
Clustering coefficient for scale free graph is: 0.4215588190762562
Average degree for scale free graph is: 196.34730538922156
```



[ ]: