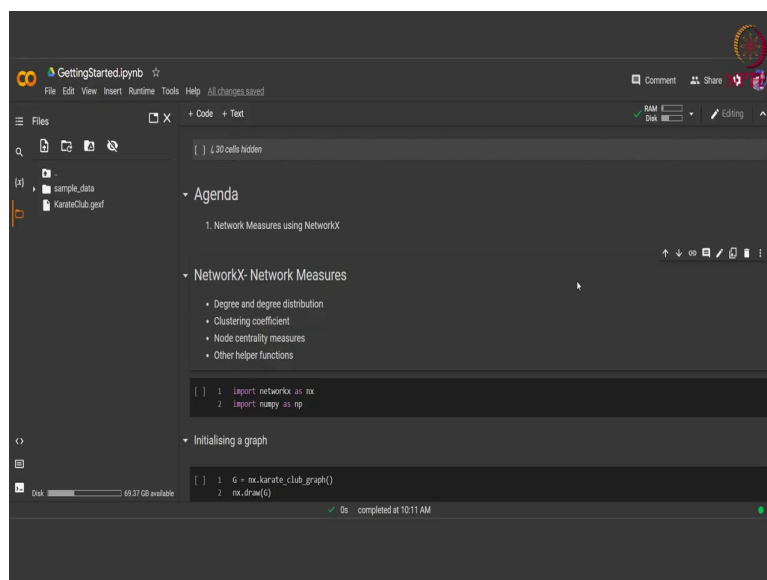


Social Network Analysis
Prof. Shivani Kumar
Department of Computer Science and Engineering
Indraprastha Institute of Information Technology, Delhi

Lecture - 12
Tutorial 3: Introduction to NetworkX - Part II

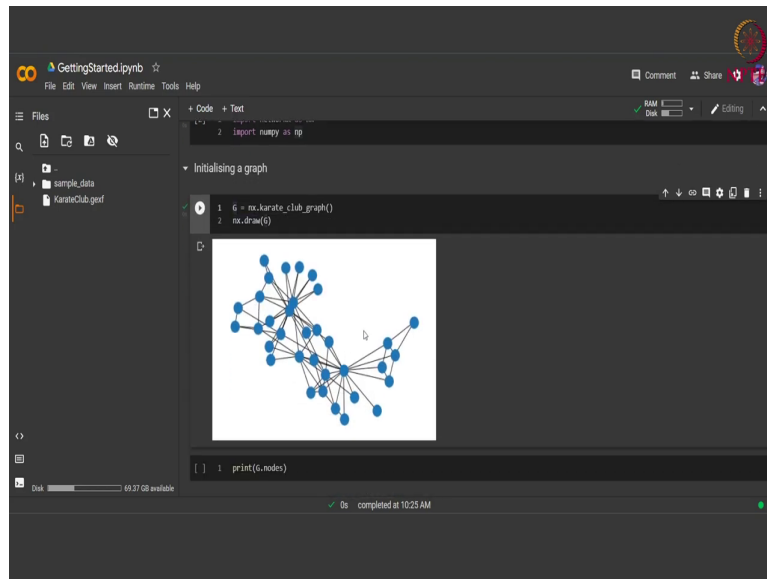
Hello everyone, welcome back to the tutorials associated with Social Network Analysis. So, we have already seen how to get started with Python and Google Collab, we have also seen how to get started with NetworkX. Today we will we will be diving deep into the NetworkX library and will be seeing some of the network measures and how we can analyze networks into different ways using the NetworkX library.

(Refer Slide Time: 00:52)



So, what is a network? Basically any network is defined by the nodes and edges that it contains right so, but to analyze the different properties of the network, we need to analyze the different type of connectivity between these nodes and edges. In order to do that in today's tutorial we will be taking a sample undirected unweighted network and will be just tweaking its properties and looking at what kind of attribute it contains in order to learn how we can use NetworkX to analyze the graph.

(Refer Slide Time: 01:35)

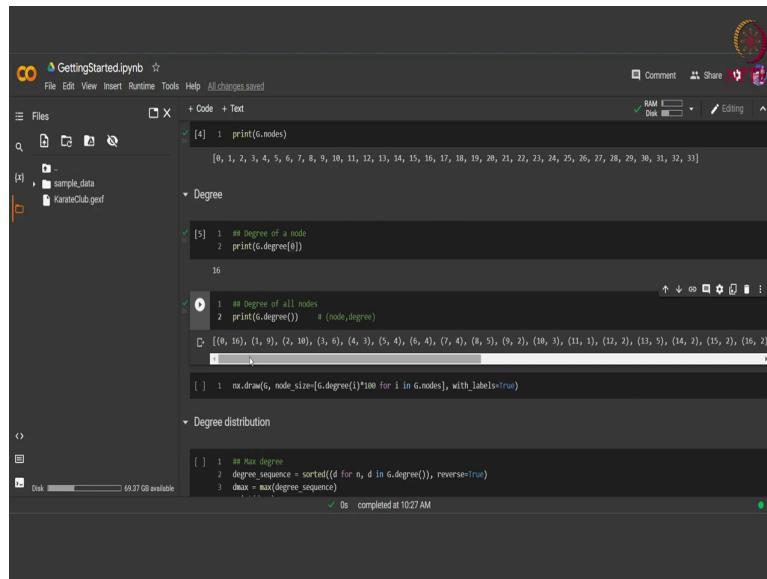


So, the NetworkX library it provides us with a lot of sample graphs already present in it we will use one such graphs known as the karate club graph. So, in this graph basically we have 34 nodes, where each node is associated with a certain club. So, there are total of two clubs in the network and each node is associated with either of the one club then there are 78 edges between these nodes and each of the edge represent whether the two nodes have a tie between them that is whether the two people have interacted in the feud in the past.

So, the basic thing we will do first we will just import the NetworkX library, we will also import the numpy library because we will be using it further we have imported the NetworkX library with as an alias nx so, that it is easy to use just like we did in the previous tutorials then in order to initialize a unweighted undirected graph with the karate club edges and nodes.

We just simply call this karate club graph function of the NetworkX library and assign this graph to an object G then we in order to just visualize the graph we call the draw function of the NetworkX library and we visualize the graph. We can see that there are some 34 nodes with certain connections between them.

(Refer Slide Time: 03:16)



```
GettingStarted.ipynb
File Edit View Insert Runtime Tools Help All changes saved
RAM 100%
Disk 100%
Editing

[4]: 1 print(G.nodes)
      [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33]

Degree
[5]: 1 # Degree of a node
      2 print(G.degree[0])
      16

[6]: 1 # Degree of all nodes
      2 print(G.degree()) # (node, degree)
      [(0, 16), (1, 9), (2, 10), (3, 6), (4, 3), (5, 4), (6, 4), (7, 4), (8, 5), (9, 2), (10, 3), (11, 1), (12, 2), (13, 5), (14, 2), (15, 2), (16, 2), (17, 1), (18, 1), (19, 1), (20, 1), (21, 1), (22, 1), (23, 1), (24, 1), (25, 1), (26, 1), (27, 1), (28, 1), (29, 1), (30, 1), (31, 1), (32, 1), (33, 1)]

[7]: 1 nx.draw(G, node_size=[G.degree(i)*100 for i in G.nodes], with_labels=True)

Degree distribution
[8]: 1 # Max degree
      2 degree_sequence = sorted((d for n, d in G.degree()), reverse=True)
      3 dmax = max(degree_sequence)
```

Let us verify whether they are 34 nodes or not and yes as said there are 34 nodes ids from 0 to 33. Now the most basic thing about a node in the network is its degree. So, a node degree is basically the number of edges incident on it right that is the number of connections it has with the other nodes or you can also say that degree is the number of one hop neighbours of a node.

So; obviously, in this undirected graph. So, in order to look at a degree of for example, say the 0th node of our network, we simply call this function on our graph object that is G, we call this degree function and tell it what which nodes information we want. So, for the 0th node we pass G dot degree 0 and we run this and we see that the node 0th degree is 16 that is there are 16 edges that are incident on the node 0.

Now in order to see the degree of all the nodes in the graph instead of like just looking at each node one by one, we can simply call this G dot degree function without any parameter inside it. So, we run this and we get a list of tuples. Now here each tuple basically each pair it represents the node and its corresponding degree. So, we have for the 0th node we have degree 16 for the first node we have the degree 9 and so, on.

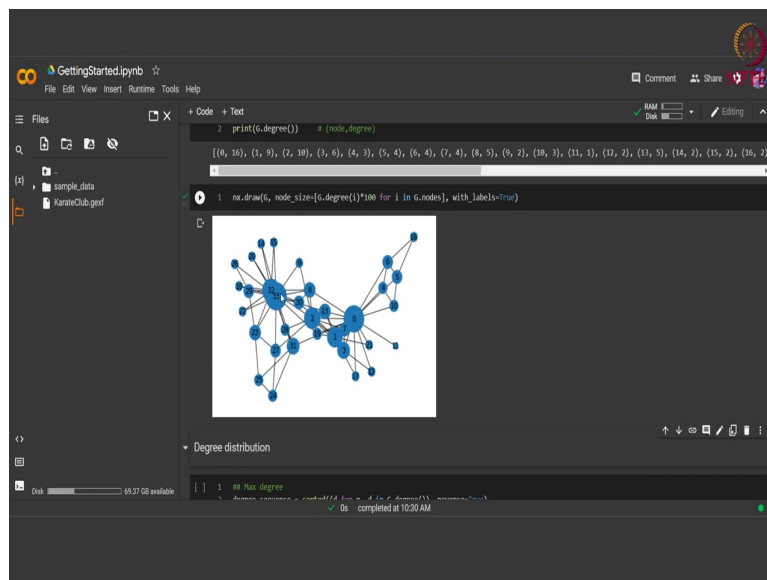
Now, one interesting thing to observe here is that although we can see the degree associated with each node in this list, but in order to visualize the graph based on this degree that is we might want that a node which has a higher degree appear big in the network whereas, a node with low degree appears smaller as compared to a higher degree node.

So, then this NetworkX library although it provides only the basic functionalities for visualization it does provide an interesting attribute inside this nx dot draw function. We can provide it with a attribute called node size and in this node size we need to pass a list of the node sizes associated with each node that is present in the graph.

So, here we just simply scale the size of each node based on the degree value of the node. So, this part of the function call is basically just scaling each of the just scaling the size of the node based on its degree. So, here we just call this G dot degree function for all the i nodes in the graph and just since the degree is small. So, for example, we have 16, 9, 10, 6 these are the degrees that we have, we need to just scale it a bit up.

So, that the graph is visible to us. So, we just multiply each of these values with a 100 so, that all of the sizes are scaled up by 100. So, we can see the graph clearly and the other attribute that we provide in this nx dot draw function is with labels that is we will be able to see the graph with the nodes in proportion to the size of the value of the degree. And the labels that is the id of the node that is 0, 1, 2, 3 that will be printed on the node.

(Refer Slide Time: 07:38)



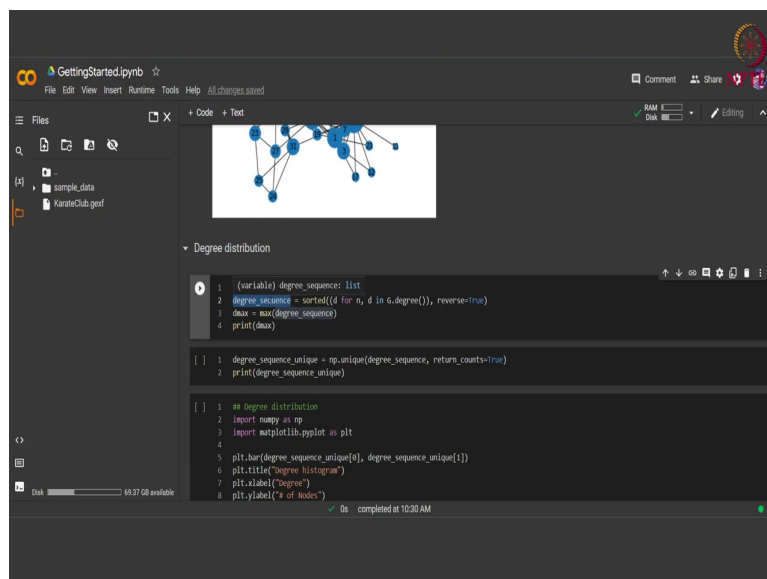
So, we just run this cell and we see that node 33, 32, 0 2 they appear bigger in size as compared to these other nodes like 16, 15, 14 which are the periphery nodes with lesser degree. So, this as you can see that the node 0 has 6 the degree 16 which is on the higher side of all the degrees that are present in the graph. So, 0 appears bigger. So, for 33 the degree is

also higher. So, let us just see what is the degree of 33. So, the degree of 33 is 17 and as expected since 17 is greater than 16. So, 33 is appear even bigger than 0.

So, by visualizing the network like this we can just simply look and tell which of the nodes has a higher degree and which of them has a lower degree. Now this one this was for the undirected graph. Now for a directed graph a degree can be of two types that is in degree and out degree. So, in degree is basically the number of incoming edges towards the node and out degree would be simply the number of outgoing edges from the node right.

So, in this in today's tutorial we will restrict our self to undirected graph so, that it is like we will be able to cover more topics. So, moving forward. So, we saw the degree of each node and the all the nodes in the present in the graph we might also need to plot the degree distribution of the network.

(Refer Slide Time: 09:24)



The screenshot shows a Jupyter Notebook interface with a dark theme. At the top, the file name is 'GettingStarted.ipynb'. The left sidebar shows a file explorer with 'sample_data' and 'KarateClub.gexf'. The main area is split into two parts: a top part showing a network graph with blue nodes and edges, and a bottom part showing Python code for calculating the degree distribution. The code includes comments and prints the degree sequence, maximum degree, and a histogram.

```
1 (variable) degree_sequence: list
2 degree_sequence = sorted(d for n, d in G.degree(), reverse=True)
3 dmax = max(degree_sequence)
4 print(dmax)

[] 1 degree_sequence_unique = np.unique(degree_sequence, return_counts=True)
   2 print(degree_sequence_unique)

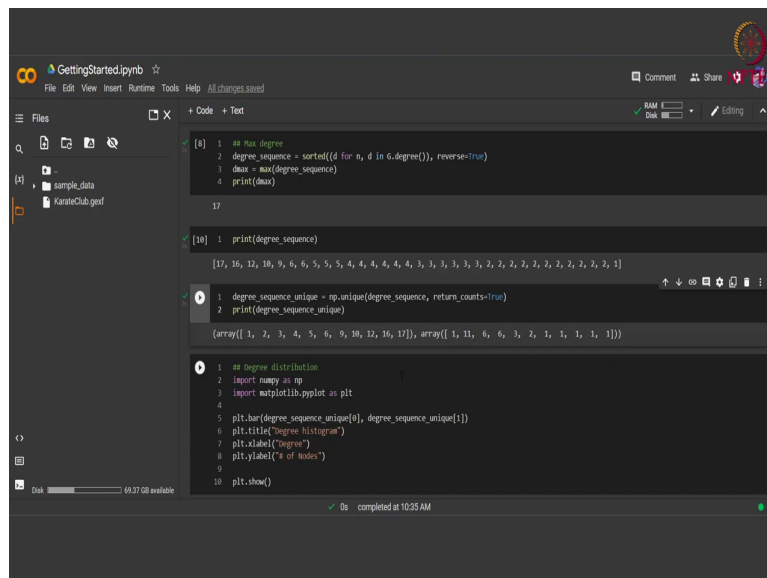
[] 1 # degree distribution
   2 import numpy as np
   3 import matplotlib.pyplot as plt
   4
   5 plt.bar(degree_sequence_unique[0], degree_sequence_unique[1])
   6 plt.title("degree histogram")
   7 plt.xlabel("degree")
   8 plt.ylabel("# of nodes")
```

So, the degree distribution it is essential to compute sometimes to just compare different kinds of network. So, as we already discussed in the theoretical section that a real world graph it follows a power law in its degree distribution right. So, we can see such a distribution or such a pattern by plotting the degrees in a histogram fashion.

So, let us just try to do that using NetworkX. So, first what we will do? So, let us just first calculate the maximum degree that is present in our network. So, we just capture the whole

degree sequence that is the like all the degrees of all the nodes in this degree sequence variable.

(Refer Slide Time: 10:15)



```
GettingStarted.ipynb
File Edit View Insert Runtime Tools Help All changes saved
RAM 100%
Disk 100%
Editing

Files
sample_data
KarateClub.gesf

[8]: 1 #! Max degree
      2 degree_sequence = sorted((d for n, d in G.degree()), reverse=True)
      3 dmax = max(degree_sequence)
      4 print(dmax)

17

[10]: 1 print(degree_sequence)

[17, 16, 12, 10, 9, 6, 6, 5, 5, 5, 4, 4, 4, 4, 4, 4, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1]

[11]: 1 degree_sequence_unique = np.unique(degree_sequence, return_counts=True)
      2 print(degree_sequence_unique)

(array([ 1,  2,  3,  4,  5,  6,  9, 10, 12, 16, 17]), array([ 1, 11,  6,  6,  3,  2,  1,  1,  1,  1]))

[12]: 1 #! Degree distribution
      2 import numpy as np
      3 import matplotlib.pyplot as plt
      4
      5 plt.bar(degree_sequence_unique[0], degree_sequence_unique[1])
      6 plt.title("Degree histogram")
      7 plt.xlabel("Degree")
      8 plt.ylabel("# of nodes")
      9
      10 plt.show()
```

How do we do that? We just call this degree G dot degree function which gives us a list of tuple as we already saw. So, that is a list of tuple that is these n and d will capture the node and its corresponding degree and for each degree that is there we simply put it in another tuple just the degree part of it.

And then we sort it in a reverse manner and we take the maximum of this sequence. So, and now we just print the max. So, let us just see yeah. So, we are getting the maximum as 17 which we already saw above that node 33 had the degree 17 and we are getting the maximum as 17. So, now, what we will do is in order to plot the degree distribution what we need is basically the unique values of the degree and how many times a node obtains this degree right.

So, in this degree sequence which is like. So, let us just print this degree sequence and you will be able to see what is it actually contains. So, let me just print it in a nice fashion yeah. So, it is basically a list of all the degrees that is present in the network. So, we see that the degree 17 is present once, degree 16 is present once and so, is the case for 12, 10 and 9, but the degree 6 is present twice that is two nodes of the network has the degree 6.

And the degree 5 comes 3 times, then the degree 4 comes the number of times then 3 2 and 1. So, basically a degree distribution graph should show us the probability that if we consider a random node from the network what will be the probability of it to have some degree right? So, in order to do that we need to count the degree values. So, we call this np dot unique function for this.

So, what it does? It will return us with this with basically a set of the unique values that is present in this list. So, for example, 17, 16, 12, 10, 9 then 6 ones and 5, 4, 3, 2 and 1 these are the unique values that are present in this list, but we also want that how many times these value comes. So, we also pass this attribute that is return counts equal to true.

So, now we just see [FL] how does this degree sequence unique looks like. So, it is basically a tuple of two arrays where the first array represents the unique values that we have that is 1, 2, 3, 4, 5, 6, 9, 10, 12, 16 and 17 that is the unique degree values and the next array tells us the count for each of the corresponding values that is the degree 1 occurs once, degree 2 occurs 11 times, 3 occurs 6 and so, on.

(Refer Slide Time: 13:34)

```

17
[10]: print(degree_sequence)
[17, 16, 12, 10, 9, 6, 5, 5, 5, 4, 4, 4, 4, 4, 4, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1]

[11]: degree_sequence_unique = np.unique(degree_sequence, return_counts=True)
      print(degree_sequence_unique)
(array([ 1,  2,  3,  4,  5,  6,  9, 10, 12, 16, 17]), array([ 1, 11,  6,  3,  2,  1,  1,  1,  1]))

0: # Degree distribution
1: import numpy as np
2: import matplotlib.pyplot as plt
3:
4: tuple: degree_sequence_unique
   (2 items) (ndarray with shape (11,)) (ndarray with shape (11,))
5: plt.bar(degree_sequence_unique[0], degree_sequence_unique[1])
6: plt.title("Degree Histogram")
7: plt.xlabel("degree")
8: plt.ylabel("# of nodes")
9:
10: plt.show()

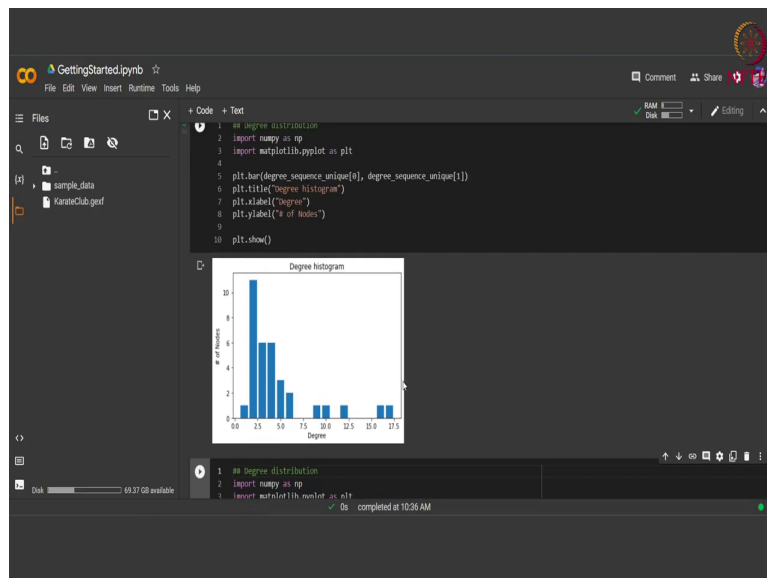
[ ]: # Degree distribution
1: import numpy as np
2: import matplotlib.pyplot as plt
3:
4:
5:
6:
7:
8:
9:
10:
11:
12:
13:
14:
15:
16:
17:
18:
19:
20:
21:
22:
23:
24:
25:
26:
27:
28:
29:
30:
31:
32:
33:
34:
35:
36:
37:
38:
39:
40:
41:
42:
43:
44:
45:
46:
47:
48:
49:
50:
51:
52:
53:
54:
55:
56:
57:
58:
59:
60:
61:
62:
63:
64:
65:
66:
67:
68:
69:
70:
71:
72:
73:
74:
75:
76:
77:
78:
79:
80:
81:
82:
83:
84:
85:
86:
87:
88:
89:
90:
91:
92:
93:
94:
95:
96:
97:
98:
99:
100:
101:
102:
103:
104:
105:
106:
107:
108:
109:
110:
111:
112:
113:
114:
115:
116:
117:
118:
119:
120:
121:
122:
123:
124:
125:
126:
127:
128:
129:
130:
131:
132:
133:
134:
135:
136:
137:
138:
139:
140:
141:
142:
143:
144:
145:
146:
147:
148:
149:
150:
151:
152:
153:
154:
155:
156:
157:
158:
159:
160:
161:
162:
163:
164:
165:
166:
167:
168:
169:
170:
171:
172:
173:
174:
175:
176:
177:
178:
179:
180:
181:
182:
183:
184:
185:
186:
187:
188:
189:
190:
191:
192:
193:
194:
195:
196:
197:
198:
199:
200:
201:
202:
203:
204:
205:
206:
207:
208:
209:
210:
211:
212:
213:
214:
215:
216:
217:
218:
219:
220:
221:
222:
223:
224:
225:
226:
227:
228:
229:
230:
231:
232:
233:
234:
235:
236:
237:
238:
239:
240:
241:
242:
243:
244:
245:
246:
247:
248:
249:
250:
251:
252:
253:
254:
255:
256:
257:
258:
259:
260:
261:
262:
263:
264:
265:
266:
267:
268:
269:
270:
271:
272:
273:
274:
275:
276:
277:
278:
279:
280:
281:
282:
283:
284:
285:
286:
287:
288:
289:
290:
291:
292:
293:
294:
295:
296:
297:
298:
299:
300:
301:
302:
303:
304:
305:
306:
307:
308:
309:
310:
311:
312:
313:
314:
315:
316:
317:
318:
319:
320:
321:
322:
323:
324:
325:
326:
327:
328:
329:
330:
331:
332:
333:
334:
335:
336:
337:
338:
339:
340:
341:
342:
343:
344:
345:
346:
347:
348:
349:
350:
351:
352:
353:
354:
355:
356:
357:
358:
359:
360:
361:
362:
363:
364:
365:
366:
367:
368:
369:
370:
371:
372:
373:
374:
375:
376:
377:
378:
379:
380:
381:
382:
383:
384:
385:
386:
387:
388:
389:
390:
391:
392:
393:
394:
395:
396:
397:
398:
399:
400:
401:
402:
403:
404:
405:
406:
407:
408:
409:
410:
411:
412:
413:
414:
415:
416:
417:
418:
419:
420:
421:
422:
423:
424:
425:
426:
427:
428:
429:
430:
431:
432:
433:
434:
435:
436:
437:
438:
439:
440:
441:
442:
443:
444:
445:
446:
447:
448:
449:
450:
451:
452:
453:
454:
455:
456:
457:
458:
459:
460:
461:
462:
463:
464:
465:
466:
467:
468:
469:
470:
471:
472:
473:
474:
475:
476:
477:
478:
479:
480:
481:
482:
483:
484:
485:
486:
487:
488:
489:
490:
491:
492:
493:
494:
495:
496:
497:
498:
499:
500:
501:
502:
503:
504:
505:
506:
507:
508:
509:
510:
511:
512:
513:
514:
515:
516:
517:
518:
519:
520:
521:
522:
523:
524:
525:
526:
527:
528:
529:
530:
531:
532:
533:
534:
535:
536:
537:
538:
539:
540:
541:
542:
543:
544:
545:
546:
547:
548:
549:
550:
551:
552:
553:
554:
555:
556:
557:
558:
559:
560:
561:
562:
563:
564:
565:
566:
567:
568:
569:
570:
571:
572:
573:
574:
575:
576:
577:
578:
579:
580:
581:
582:
583:
584:
585:
586:
587:
588:
589:
590:
591:
592:
593:
594:
595:
596:
597:
598:
599:
600:
601:
602:
603:
604:
605:
606:
607:
608:
609:
610:
611:
612:
613:
614:
615:
616:
617:
618:
619:
620:
621:
622:
623:
624:
625:
626:
627:
628:
629:
630:
631:
632:
633:
634:
635:
636:
637:
638:
639:
640:
641:
642:
643:
644:
645:
646:
647:
648:
649:
650:
651:
652:
653:
654:
655:
656:
657:
658:
659:
660:
661:
662:
663:
664:
665:
666:
667:
668:
669:
670:
671:
672:
673:
674:
675:
676:
677:
678:
679:
680:
681:
682:
683:
684:
685:
686:
687:
688:
689:
690:
691:
692:
693:
694:
695:
696:
697:
698:
699:
700:
701:
702:
703:
704:
705:
706:
707:
708:
709:
710:
711:
712:
713:
714:
715:
716:
717:
718:
719:
720:
721:
722:
723:
724:
725:
726:
727:
728:
729:
730:
731:
732:
733:
734:
735:
736:
737:
738:
739:
740:
741:
742:
743:
744:
745:
746:
747:
748:
749:
750:
751:
752:
753:
754:
755:
756:
757:
758:
759:
760:
761:
762:
763:
764:
765:
766:
767:
768:
769:
770:
771:
772:
773:
774:
775:
776:
777:
778:
779:
780:
781:
782:
783:
784:
785:
786:
787:
788:
789:
790:
791:
792:
793:
794:
795:
796:
797:
798:
799:
800:
801:
802:
803:
804:
805:
806:
807:
808:
809:
810:
811:
812:
813:
814:
815:
816:
817:
818:
819:
820:
821:
822:
823:
824:
825:
826:
827:
828:
829:
830:
831:
832:
833:
834:
835:
836:
837:
838:
839:
840:
841:
842:
843:
844:
845:
846:
847:
848:
849:
850:
851:
852:
853:
854:
855:
856:
857:
858:
859:
860:
861:
862:
863:
864:
865:
866:
867:
868:
869:
870:
871:
872:
873:
874:
875:
876:
877:
878:
879:
880:
881:
882:
883:
884:
885:
886:
887:
888:
889:
890:
891:
892:
893:
894:
895:
896:
897:
898:
899:
900:
901:
902:
903:
904:
905:
906:
907:
908:
909:
910:
911:
912:
913:
914:
915:
916:
917:
918:
919:
920:
921:
922:
923:
924:
925:
926:
927:
928:
929:
930:
931:
932:
933:
934:
935:
936:
937:
938:
939:
940:
941:
942:
943:
944:
945:
946:
947:
948:
949:
950:
951:
952:
953:
954:
955:
956:
957:
958:
959:
960:
961:
962:
963:
964:
965:
966:
967:
968:
969:
970:
971:
972:
973:
974:
975:
976:
977:
978:
979:
980:
981:
982:
983:
984:
985:
986:
987:
988:
989:
990:
991:
992:
993:
994:
995:
996:
997:
998:
999:
1000:

```

Now, we will use this tuple to create our histogram now how to do that? We will need the matplotlib library in order to just plot the bar graph. Now the x axis of the bar graph should contain the degree values and the degree value is present in the first element of this tuple. So, we simply pass the first element of the tuple as the x axis and the y axis of the degree

distribution should contain the count of the degree values or the probability we will come to that later.

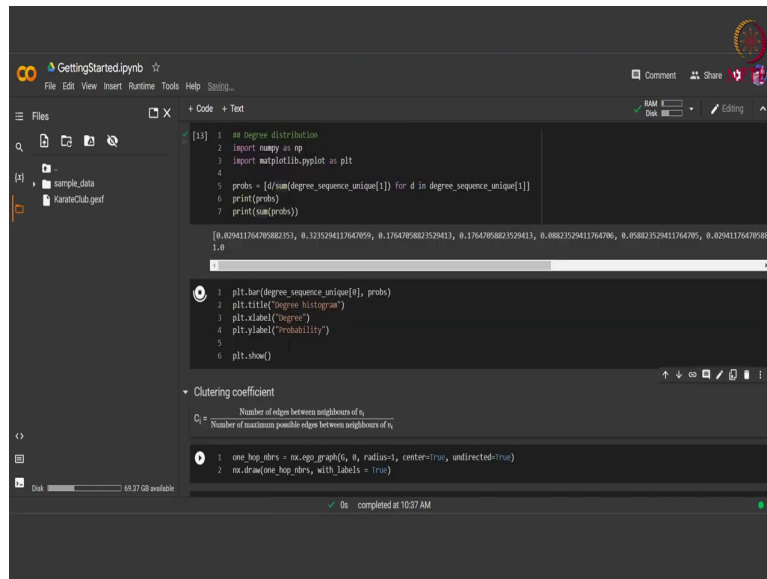
(Refer Slide Time: 14:09)



So, right now we just simply plot the count of the degree values which is present in the second element of this tuple. So, we pass the second element here we just put some title in the x axis label and y axis label and just show the network. So, this is basically our degree distribution that we are getting that is the degree 2 comes the most amount of time whereas, 3 and 4 comes equally as we saw that the both comes for 6 number of times and followed by a degree 5 6 and so, on.

Now we can instead of just you know plotting the exact values for the degree the exact number of nodes we can also plot the probability.

(Refer Slide Time: 14:57)



```
1 # Degree distribution
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 probs = [d/sum(degree_sequence_unique[1]) for d in degree_sequence_unique[1]]
6 print(probs)
7 print(sum(probs))

[0.40241176470588235, 0.3235294117647059, 0.17647058823529413, 0.17647058823529413, 0.08823529411764706, 0.058823529411764705, 0.02941176470588235]
1.0

1 plt.bar(degree_sequence_unique[0], probs)
2 plt.title("Degree histogram")
3 plt.xlabel("degree")
4 plt.ylabel("probability")
5
6 plt.show()
```

Clustering coefficient

$C_i = \frac{\text{Number of edges between neighbours of } n_i}{\text{Number of maximum possible edges between neighbours of } n_i}$

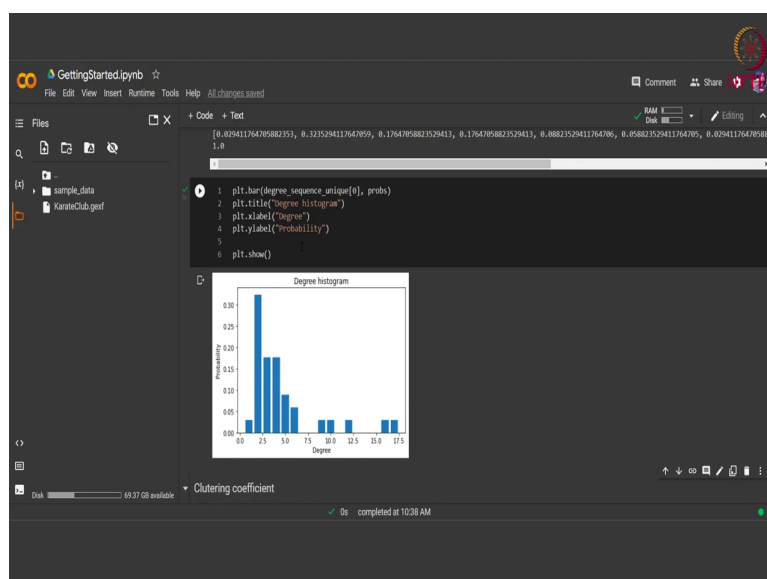
```
1 one_hop_nbrs = nx_ego_graph(G, 0, radius=1, center=True, undirected=True)
2 nx.draw(one_hop_nbrs, with_labels=True)
```

completed at 10:27 AM

So, now how to calculate the probability? So, we just divide the count by the total number of degrees total number of edges and nodes right. So, we just simply like take the sum of the of all the degree values and divide by each of the corresponding value of the like the count of the node that the degree comes in right.

But since the denominator is same the normalization factor is same, we will get the like the graph would be similar, but here since it is probability. So, the sum of all these probabilities would be 1 as expected.

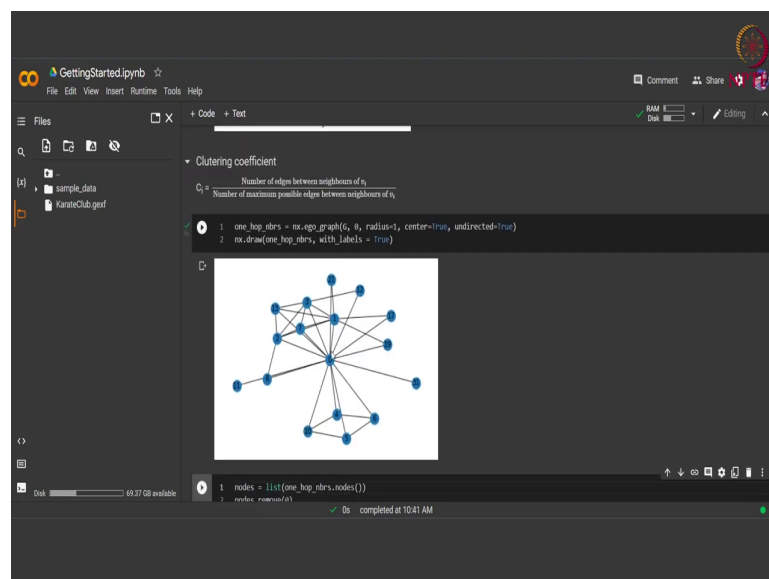
(Refer Slide Time: 15:40)



We plot this and we see that the distribution is same, but now the values for each of the each of the degree is restricted between 0 and 1 since its a probability. So, now, if we by looking at this graph we can see that if we randomly take consider a node from the network, the probability that it that the corresponding degree for that node is 2 is the highest and followed by 3, 4 and so, on.

Now apart from the degree and the degree distribution, another metric which helps us to understand the connectivity of the node or more importantly the local neighborhood of the node that is how much a nodes neighborhood is connected to like one another it is called clustering coefficient.

(Refer Slide Time: 16:44)



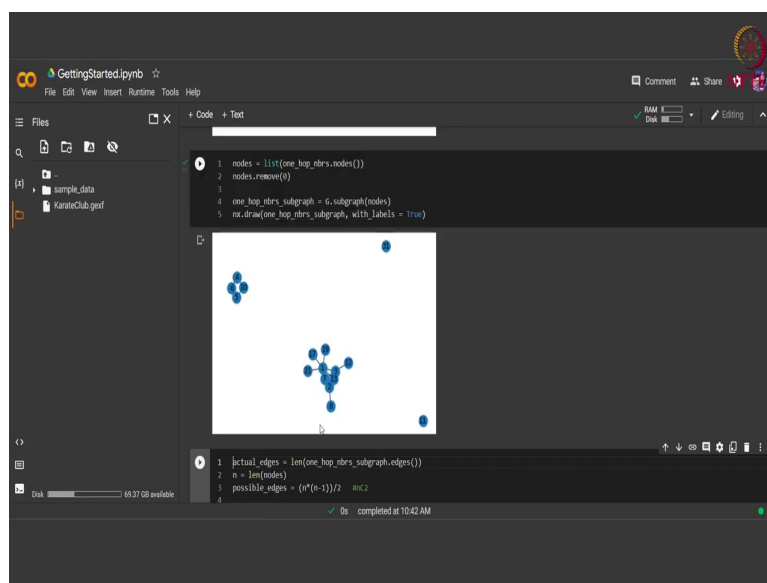
So, basically clustering coefficient what it tells is, it tells us that how much the neighbours of the node are connected. So, it uses a very simple formula that is the number of actual edges that is present between the neighbour of a node and we divide this number with the number of possible edges that there can be. So, we basically want to see that out of these possible edges how many edges do we actually have in order to see how connected are these neighbours?

So, before using the NetworkX implementation, first let us see whether the whether our intuition of the clustering coefficient matches with the values that we are getting. So, first what we will do is we will get the. So, let us like in this example let us consider the node 0 and let us try to calculate the clustering coefficient of that node.

So, the first thing would be to get the one hop neighbour sub graph, that is a graph where all the one hop neighbours of the node 0 are present the node 0 will not be present and then all the edges between the neighbours of 0 will also be there in the graph. So, this is basically called the one hop this is basically called the ego graph. So, in the ego graph the main node will be present and all the one hop neighbours of that node will be present.

So, we can get this ego graph by calling the nx dot ego graph function we pass the net the graph from which we want to get the ego graph then the node for which we want the ego net and the radius that is we just want the one hop neighbour. So, we may pass the radius as 1 that is just go one length in the path and then these are just other attributes in order to you know get the information correctly then we draw this one hop neighbours graph. So, that we just we can see how does this look like.

(Refer Slide Time: 19:02)



So, here we can see that this is the 0th node of which this ego net is. And then we have all the one hop neighbours of this node and the connections present between these neighbours ok. Just to each other not the external connection that might be present between these neighbours.

So, we have this we just create the list of all the nodes that are present in it now in order to calculate the clustering coefficient we do not need this node the main node for which the clustering coefficient is to be calculated because we are just concerned with the connectivity of the neighbour of the node.

So, we remove that node from the nodes list and what we do is now we just want the we also want to remove these edges which are present between the node 0 and the other nodes. So, either we can remove these edges one by one from this ego net or we can also call this sub graph function between all the neighbour nodes. So, what this function will do is, it will just return the nodes and the edges present between the list of nodes that we pass to this function. So, this list basically contains all the one hop neighbours of the node 0 except the node 0 of course.

So, we create this in the sub graph one hop neighbour sub graph and we draw this here. So, let us see how does this look like. So, we see we get these four components to us. So, here you can see that this 1, 7, 2, 13 3 this basically comes from this side 1 7, 13, 2, 3 and so, on this 10, 5, 4, 6 comes here and then we have this 31 and 11 two isolated components with us.

(Refer Slide Time: 21:01)

```

GettingStarted.ipynb
File Edit View Insert Runtime Tools Help
+ Code + Text
[17] 1 actual_edges = len(one_hop_nebs_subgraph.edges())
      2 n = len(nodes)
      3 possible_edges = (n*(n-1))/2 #nC2
      4
      5 print(actual_edges, possible_edges)

18 130.0

[18] 1 cc = actual_edges/possible_edges
      2 print(cc)

0.15

[19] 1 # clustering coefficient of a node
      2 print(mx.clustering(G,0))

0.15

[20] 1 # clustering coefficient of all nodes
      2 print(mx.clustering(G))

[21] 1 mx.draw(G, node_size=[mx.clustering(G,i)*1000 for i in G.nodes], with_labels=True)

Centrality Measures
Disk 69.27 GB available
0s completed at 10:45 AM

```

Then the number of actual edges that is present between these neighbours or this one hop neighbours we can get that by calling the dot edges function on this sub graph. We calculate the length of the this tuple that we get sorry we calculate the length of this tuple that we get and assign this value this length value that is the number of these edges to an actual edge variable.

Then basically the total number of nodes would be the length of node and all the possible edges would be nC_2 that is we take all the possible pairs of the nodes and if there is an edge consider an edge between them. So, we use this formula to calculate the possible edges and

the actual edges and let us just print it. So, the actual edges that are present in the node are 18 whereas, according to the number of nodes that are present here they can be a total of 120 edges that can be present here.

Now, based on the formula for clustering coefficient, we just divide this actual edges with possible edges and print the value here and we see that the clustering coefficient for the node 0 is coming out to be 0.15. Now NetworkX also provides us with the function to calculate the clustering coefficient the function is called clustering and we pass the graph and the node for which we want the clustering coefficient to this function and we print it and we see that as we calculated the clustering coefficient for node 0 is coming out to be 0.15.

(Refer Slide Time: 22:48)

The screenshot shows a Jupyter Notebook interface with the following code and output:

```

0.15

[19] 1 # clustering coefficient of a node
      2 print(nx.clustering(G,0))

0.15

[20] 1 # clustering coefficient of all nodes
      2 print(nx.clustering(G))

{0: 0.15, 1: 0.3333333333333333, 2: 0.24444444444444444, 3: 0.6666666666666666, 4: 0.6666666666666666, 5: 0.5, 6: 0.5, 7: 1.0, 8: 0.5, 9: 0, 10:
}

1 nx.draw(G, node_size=[nx.clustering(G,i)*1000 for i in G.nodes], with_labels=True)

```

Below the code, there is a section titled "Centrality Measures" containing the formula for Degree centrality:

$$C_d(v) = \frac{\text{deg}(v)}{\sum_{u \in G} \text{deg}(u)}$$

$$C_d(v) = \frac{\text{deg}(v)}{N-1}$$

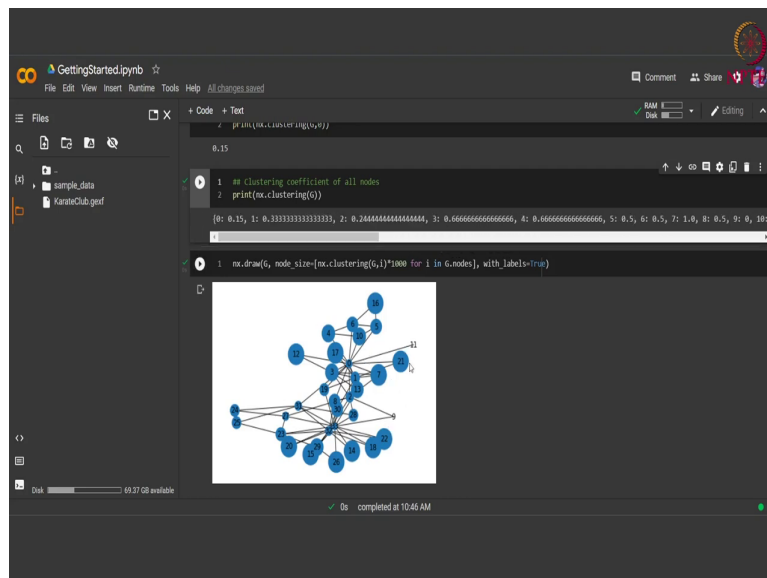
The notebook also shows a status bar at the bottom indicating "0s completed at 10:45 AM".

We can also calculate the clustering coefficient of all nodes in the graph altogether by simply calling the nx dot clustering function on the node on the whole graph G. We calculate this and we are returned with a list of the clustering coefficient where the keys are the nodes and the values are the clustering coefficient corresponding with each of these nodes. So, we see that the 0th node has a clustering coefficient of 0.15 and the 33 node has a clustering coefficient of 0.11.

And we already saw that how it is calculated intuitively. Now we can also just like we did with the degree function we can also scale the size of the nodes of a network using the clustering coefficient value. So, we calculate the clustering coefficient for each of the node

that is present in the graph and since the clustering coefficient is even smaller than the degree right since because it will lie between 0 and 1.

(Refer Slide Time: 24:00)



So, we just multiply it we just scale it up with like the value 1000 and we draw the graph and we see that the that the nodes with higher clustering coefficient are now appear to be bigger whereas, the nodes with lower clustering coefficient appearing to be smaller.

For example, here you can see 11 and 9. So, their clustering coefficient might be close to 0 right. So, here you can see that 9th clustering coefficient is 0 and 11th clustering coefficient is also 0. So, that is why this side there is no basically there is no node that can be seen for these two these two ids of the node. Now moving on from clustering coefficient. So, it clustering coefficient basically told us about the local neighbourhood of the node.

Now we might want to see how the node looks like globally in the structure of the network. So, in order to do that there is something called centrality measures. So, basically these measures they tell us how central the node is or how important the node is based on different attributes of the node. So, first let us consider the most basic attribute that we have that is the degree. So, based on the degree we can calculate the degree centrality of the node.

So, a degree centrality basically would tell us the number of connections of a node that is if a node is well connected in the network that is it has a higher number of one hop neighbours,

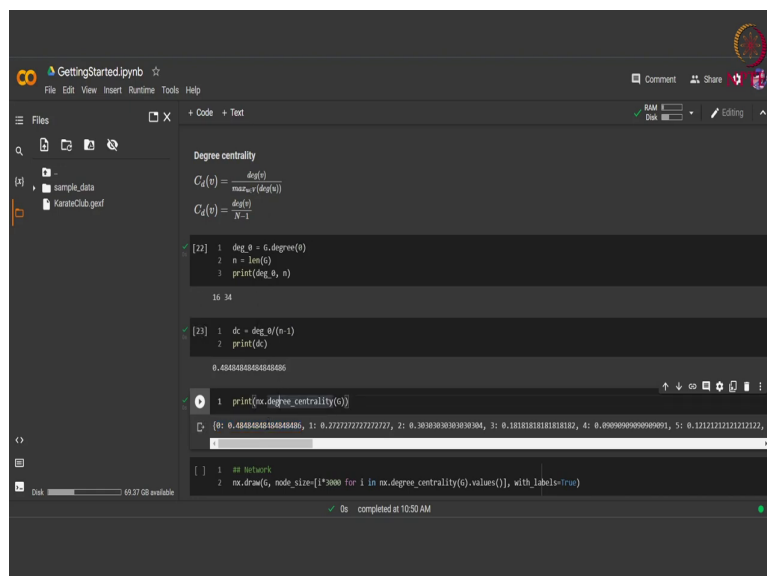
then that node will have a higher degree centrality. So, in order just. So, there can be different methods to calculate this degree centrality.

So, one of the method that we might have discussed in the class is the is when we divide the degree of the node of the concerned node for which we want to calculate the centrality value with this maximum degree that is present in the whole network right or the other way is that we simply divide the degree of the concerned node with the total number of nodes or total number minus 1.

So, the normalization factor might not matter much because the same normalization factor would be applied to all the degrees to all the nodes of the network. So, the scaling would be proportional only. So, NetworkX uses this second formulation. So, in order to get the values similar to NetworkX we will be using this formulation. So, what we will do is, we will simply first we will also in this example also we will take the instance of the node 0 and calculate the degree centrality for the node 0.

So, we first calculate the degree of the node 0 which is the numerator in any cases then what we do is we calculate the total number of nodes that are present in the graph by simply taking the length of the graph and passing it to the variable n and then we print these two values and we see that we already know that the degree of node 0 is 16 whereas, the total number of nodes there are 34. So, using this second formulation that we have we calculate the degree centrality.

(Refer Slide Time: 27:34)



```
GettingStarted.ipynb
File Edit View Insert Runtime Tools Help
+ Code + Text
RAM 100%
Disk 100%
Editing

Files
sample_data
karateClub.gexf

Degree centrality

$$C_d(v) = \frac{\text{deg}(v)}{\max_v(\text{deg}(v))}$$


$$C_d(v) = \frac{\text{deg}(v)}{n-1}$$


[22]: 1 deg_0 = G.degree(0)
      2 n = len(G)
      3 print(deg_0, n)

16 34

[23]: 1 dc = deg_0/(n-1)
      2 print(dc)

0.4705882352941176

1 print(nx.degree_centrality(G))

{0: 0.4705882352941176, 1: 0.2727272727272727, 2: 0.36363636363636364, 3: 0.18181818181818182, 4: 0.09090909090909091, 5: 0.12121212121212122, ...}

[ ]: 1 # nx.draw
      2 nx.draw(G, node_size=[i*3000 for i in nx.degree_centrality(G).values()], with_labels=True)
```

We say the numerator is simply the degree of the node 0 and the denominator is the is n minus 1 that is total number of nodes minus 1 and we print this degree centrality. We see that we are getting the value as 4848 and the NetworkX also provides us with the degree centrality measure we by simply calling this function degree centrality and passing it the graph we calculate this and we see that the degree centrality for node 0 is actually 0.4848 which we already calculated by using our intuition.

Now this like discussed above like clustering coefficient this also gives us a dictionary of values where the keys are the nodes and the values are the corresponding degree centrality measures.

(Refer Slide Time: 28:34)

The screenshot shows a Jupyter Notebook with the following code and output:

```

n = len(G)
print(deg_0, n)

[23] 1 dc = deg_0/(n-1)
      2 print(dc)
      0.48484848484848485

[24] 1 print(nx.degree_centrality(G))
      0.0000000000000001, 29: 0.12121212121212122, 30: 0.12121212121212122, 31: 0.18181818181818182, 32: 0.36363636363636365, 33: 0.5151515151515151

nx.NetworkX
2 nx.draw(G, node_size=[]*3000 for i in nx.degree_centrality(G).values(), with_labels=True)

```

Below the code, there is a section for Closeness centrality with the formula:

$$C_c(v) = \frac{n-1}{\sum_{u \in V} d_{uv}}$$

At the bottom, there is a small code snippet:

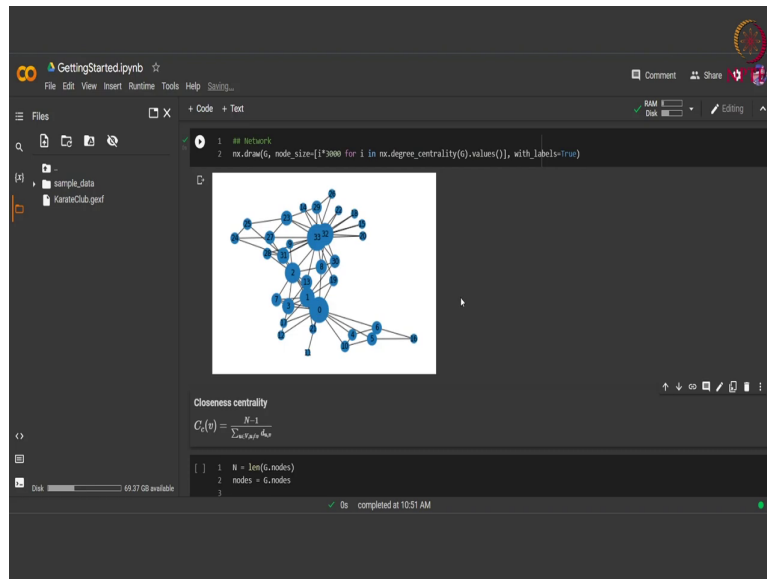
```

[] 1 n = len(G.nodes)
    2 nodes = G.nodes
    3
    4 sum_deg_0 = 0
    5 for n in nodes:

```

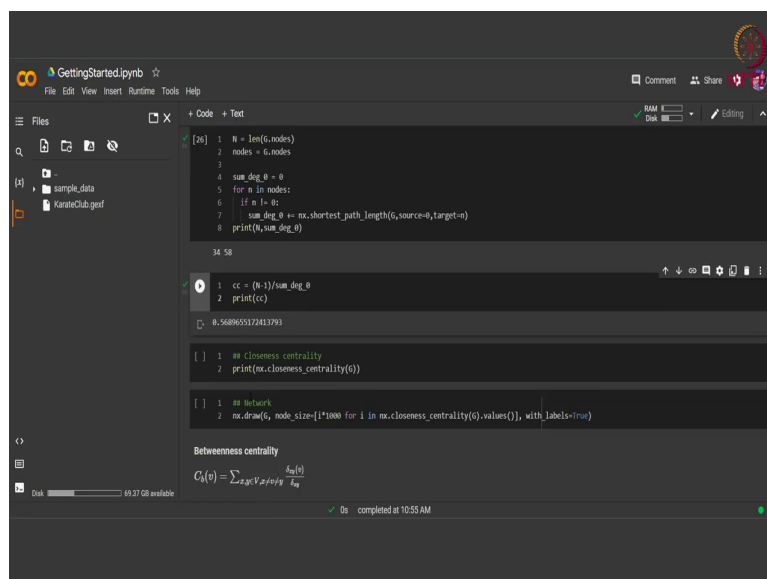
So, we can see that since we know that 0 and 33 has the higher degree. So, 0 contains the value of 0.4848 whereas, we can see all of the other values are less and for of the node 33 we are we are getting the value of 0.5151 since the degree of 33 was 17 which was more than the degree of node 0 which was 16.

(Refer Slide Time: 29:04)



Again we simply scale the size of the nodes based on the degree centrality and it is quite similar to what we were getting when we were scaling the graph based on the degree values. So, again 0 and 33 they have comparable sizes with 33 being slightly bigger than 0. Now while degree centrality told us like it indicated towards the measure of how locally connected the node is that is how many immediate neighbours it has there is another thing known as closeness centrality.

(Refer Slide Time: 29:34)



That is if we look beyond the one hop neighbours then how close a particular node is to the other nodes of the graph. So, closeness centrality basically tells us about the reach of a node. So, this is basically the formula that we can use that is the total number of nodes minus 1 divided by the distance that is the distance between the 2 nodes u and v, the shortest distance between the two nodes u and v whereas, v is basically the node for which we want to calculate the closeness centrality.

Now we take the reciprocal of this that is the distance comes in the denominator and the normalization factor that is N minus 1 comes in the numerator so, that the higher the value of the centrality the more close the node is in the network. The this is simply the reason we just take the reciprocal of this and yeah.

So, based on this formula we calculate it from scratch first. So, in order to get this N we just simply calculate the length of the nodes that is the number of total nodes that are present in the graph and we take the all these nodes in a variable called nodes.

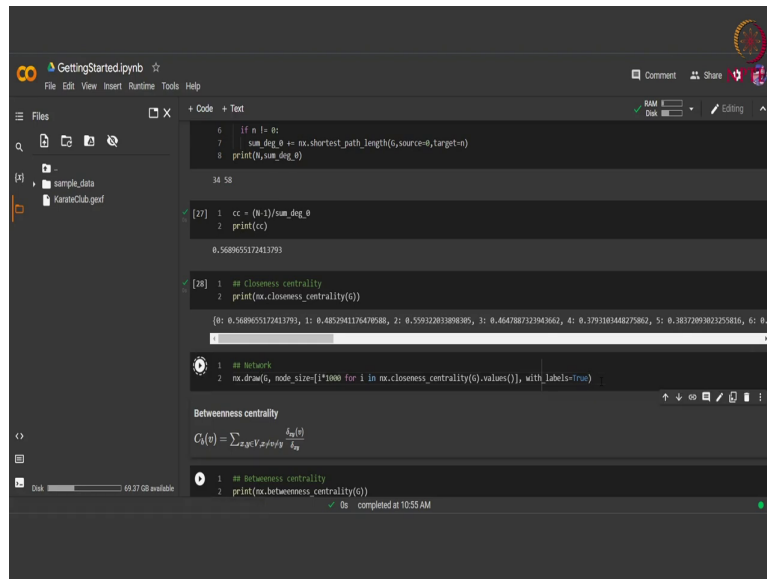
Now for all these nodes we calculate the shortest path length from the source 0 since here also we are taking the example of the node 0 and calculating the closeness centrality of that node. So, we take the source as 0 that is we need to calculate the shortest distance from 0 to all the other nodes in the network. So, we take the source as 0 and the target keeps changing based on the based on all the other nodes that are present in the network.

So, we do not calculate like the self loop kind of thing that is we do not calculate the distance between 0 and 0 we will calculate the distance between 0 1 0 2 till 0 33. So, we calculate the shortest path length by calling the nx dot shortest path length function pass it with the graph the source node at a 0 and the target node which will be taking different values.

And we simply add this distance to this variable called sum of degree 0. So, to get the denominator this whole sum to get this sum we just do this. Now we print this value of n and this distance. So, n is 34 we know there are 34 nodes in the network and the distance like the sum of all the shortest distance from 0 to all the other nodes is coming out to be 58. So, we print this.

And we use the for formulation that we already have to calculate the closeness centrality. We print this and we see that the value of the closeness centrality for the node 0 is coming out to be 0.5689.

(Refer Slide Time: 33:01)



```
GettingStarted.ipynb
File Edit View Insert Runtime Tools Help
Code + Text
6 if n != 0:
7     sum_deg_0 += nx.shortest_path_length(G,source=0,target=n)
8     print(n,sum_deg_0)
34 58
[27] 1 cc = (n-1)/sum_deg_0
2     print(cc)
0.568955172413793
[28] 1 nx.closeness_centrality
2     print(nx.closeness_centrality(G))
{0: 0.568955172413793, 1: 0.4852941176470588, 2: 0.559322033896305, 3: 0.464788732943662, 4: 0.3793103448275862, 5: 0.3837209302525816, 6: 0.1...
```

Betweenness centrality

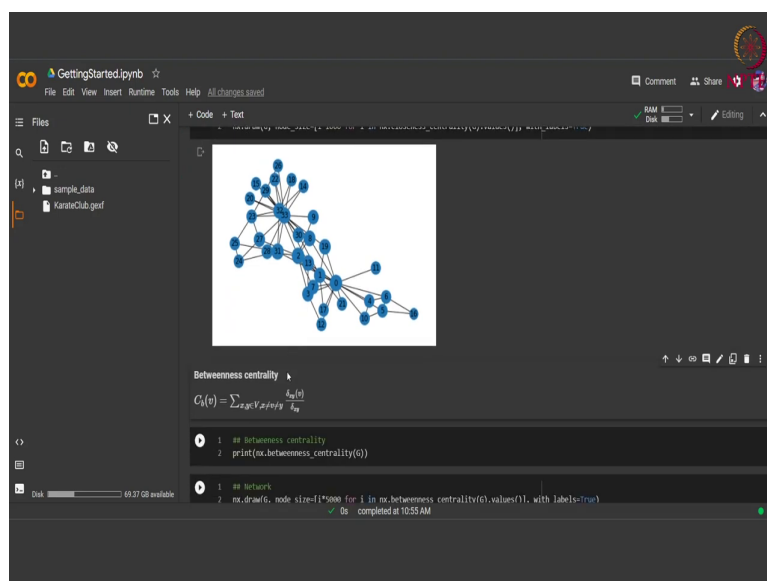
$$C_b(v) = \sum_{s,t \in V, s \neq t, v \neq s, v \neq t} \frac{A_{st}(v)}{A_{st}}$$

```
1 nx.betweenness_centrality
2 print(nx.betweenness_centrality(G))
0s completed at 10:55 AM
```

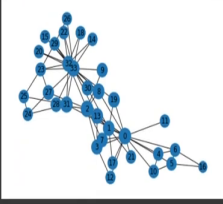
We use the inbuilt NetworkX function to calculate the closeness centrality of the whole graph and we see for the node 0 we are indeed getting this value that is 0.5689 and yeah.

So, we and like the degree centrality or the clustering coefficient we are provided with a dictionary where the key are the nodes and the value is the corresponding closeness centrality value. Again we just scale the graph based on the closeness centrality values and we see that almost all the nodes are getting a similar value for closeness centrality.

(Refer Slide Time: 33:35)



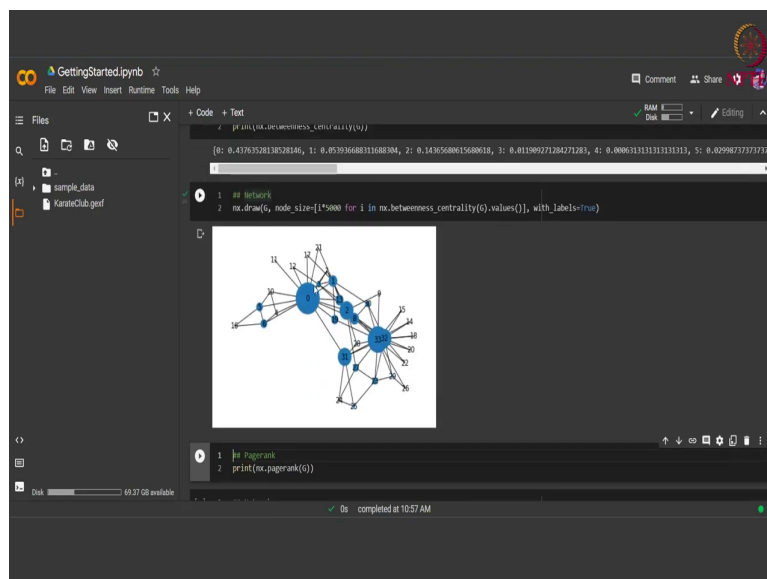
```
GettingStarted.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Code + Text
Betweenness centrality
C_b(v) = \sum_{s,t \in V, s \neq t, v \neq s, v \neq t} \frac{A_{st}(v)}{A_{st}}
1 nx.betweenness_centrality
2 print(nx.betweenness_centrality(G))
0s completed at 10:55 AM
1 nx.betweenness_centrality
2 nx.draw(G, node_size=[*1000 for i in nx.betweenness_centrality(G).values()], with_labels=True)
```



This is because the graph is a connected component in itself and all the nodes are basically reachable from all other nodes and if the connections of the graph are strong that is most of the nodes are connected to the most of the other nodes or not directly connected, but may be reachable. So, that is why the closeness centrality is almost similar for all the nodes.

Then comes the betweenness centrality. So, betweenness centrality basically what it tells is that, how many times a particular node need to be passed in order to get from one node to another node. So, that is how many times a particular node comes in between the shortest path between any other two nodes right. So, we this is the formula basically that is the number of times a particular node appears in the shortest path of the node divided by the total number of shortest path present between the pair of node. So, yeah.

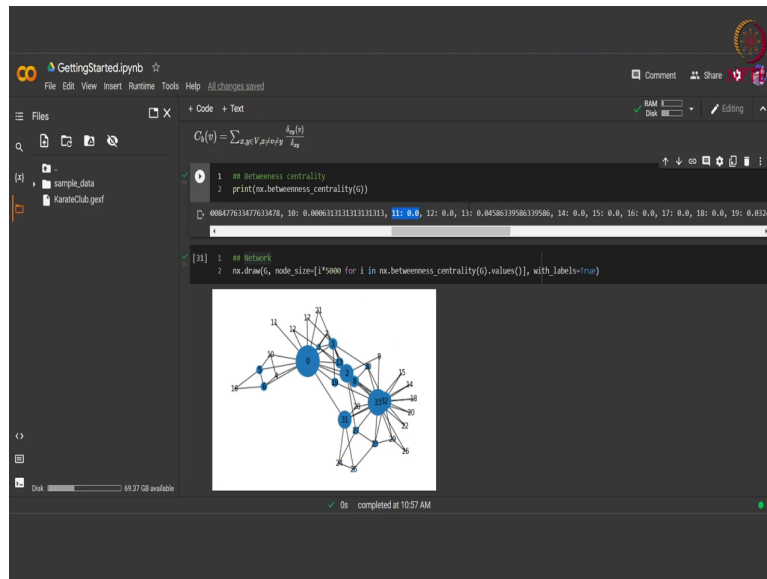
(Refer Slide Time: 34:57)



So, we simply here just see the NetworkX implementation to save our time and we see that we can call the betweenness centrality function from the NetworkX library and pass it with the graph G and it will also return a dictionary where the key are the nodes of the graph and the values the corresponding betweenness centrality values.

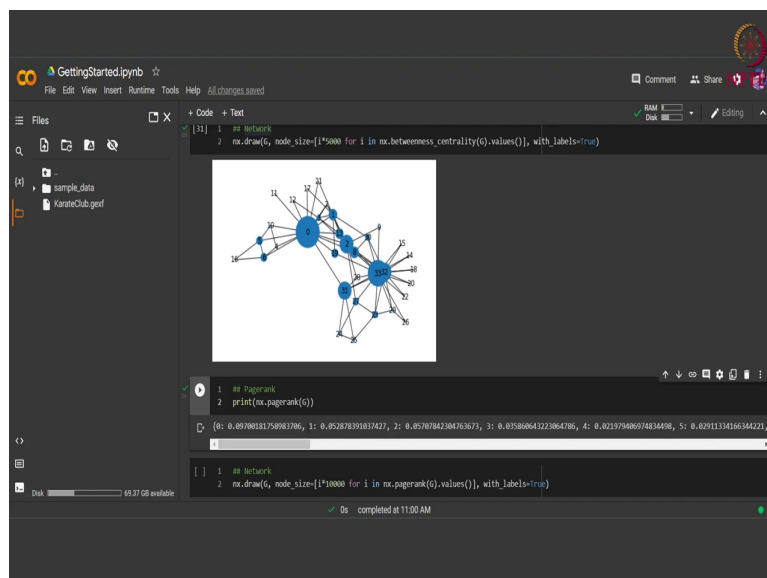
Again we can scale our the size of the nodes based on the betweenness centrality values and we see that the node 0 and 33 has a higher betweenness centrality whereas, all the periphery nodes they have betweenness centrality close to 0 24 and 25 might have some betweenness centrality, but these 11, 12, 17, 21 these are basically 0.

(Refer Slide Time: 35:53)



So, let us just verify once the closeness centrality for 11 is indeed coming out to be the betweenness centrality for 11 is indeed coming out to be 0. For 12 also it is 0, for 14, 15, 16, 17, 18 all of these are 0 and as can be seen from the visualization of the graph they are invisible that is since the betweenness centrality is coming out to be 0.

(Refer Slide Time: 36:24)



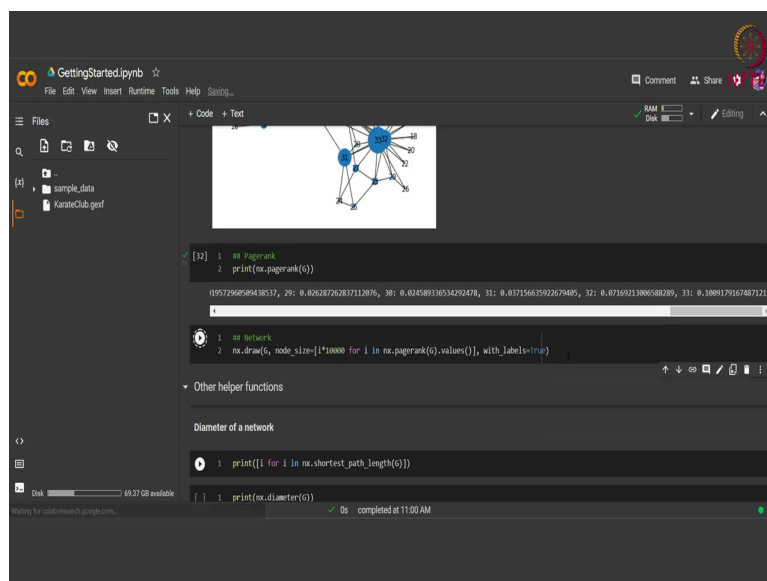
Then another interesting centrality measure or another interesting measure for to calculate the importance of a node is pagerank. So, page rank is basically a mechanism that was suggested

by Larry page of course, a co-founder of Google and it was basically used to rank the web pages of a search result right.

So, here in the domain of networks we can use this formula to calculate a ranking or the to gauge the importance of all the nodes that are present in the networks. So, if we consider like the nodes to be web pages, then the importance of a node would be like to the importance of a web page. Now this is a recursive form recursive algorithm that is the importance of a particular node will depend on the importance of its neighbours.

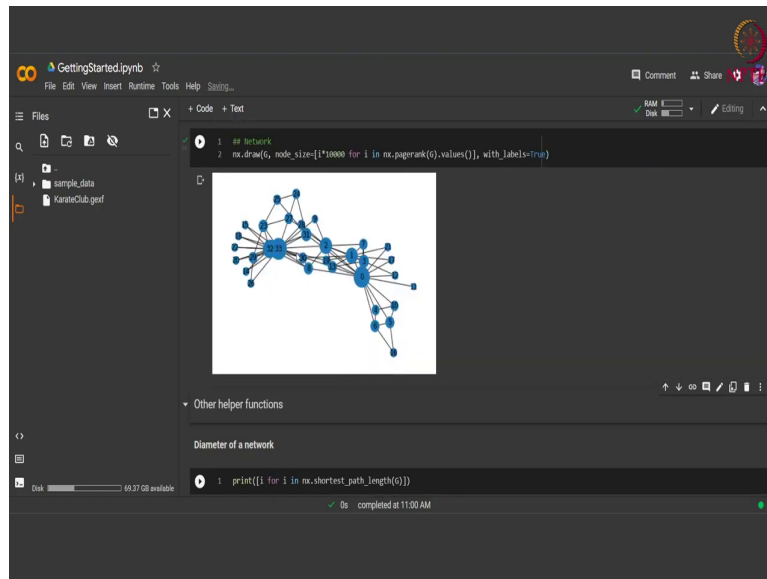
So, we I will not go into the depth of the of the algorithm, but I will focus on the fact that NetworkX provides us with the calculation of the pagerank of each node by simply a very simple one line function call. So, we call this pagerank function on the graph G and we see that it returns us with a pagerank value that is a prestige value or a value of importance for each node that is we can see that the node 0 has the page rank value of 0.097 and whereas, the page rank for 1 2 and 3 are smaller.

(Refer Slide Time: 38:10)



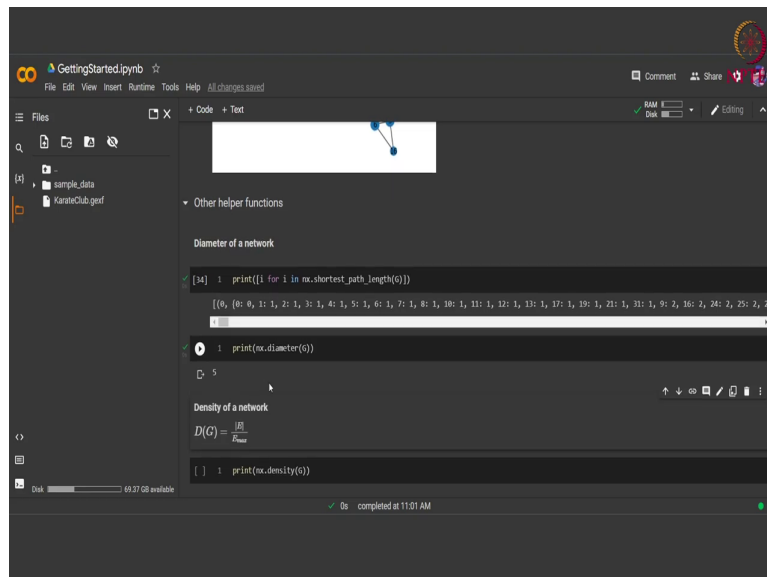
And the node 33 which was highly connected to the nodes has the highest page rank value of 0.1009; we can again just scale the size of the node based on the page rank value.

(Refer Slide Time: 38:25)



And we see that the node 0 and 33 are bigger in size since they have a higher page rank value.

(Refer Slide Time: 38:31)



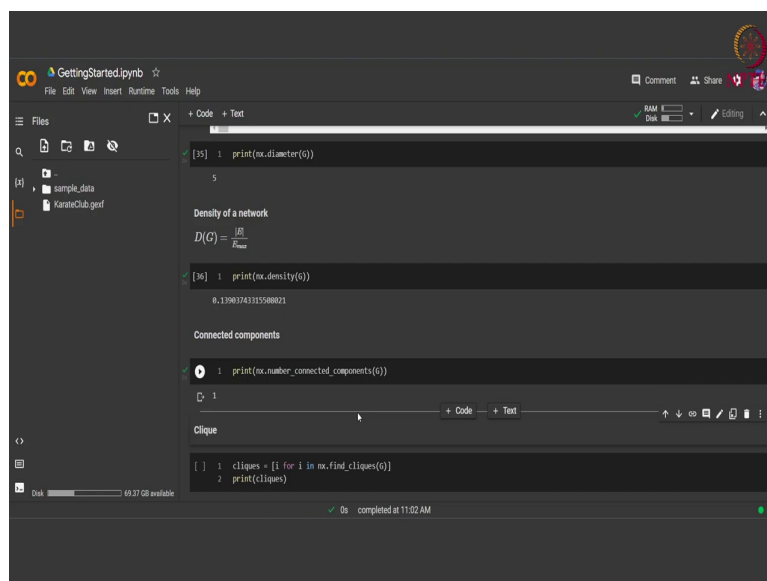
And we have other apart from these centrality measures we can also have other helper functions that is we might want to see the diameter of the network.

So, basically a diameter of the network is the longest shortest path that we have that is if we calculate all the shortest path between all pairs of nodes. So, the farthest distance between

any pair of node that would be the diameter of our network. So, first we just see all the shortest path between all the nodes. So, for the node 0 we have the path between node 0 and 0 as 0 then 0 and 1 as 1.

Similarly, for 0 and 2 as 1 and so, on we get this tuple of dictionaries. Now in order to calculate the diameter we call the NetworkX function diameter and pass the graph G to it and we see that the longest distance between any two pair in our network is 5. So, the diameter of our network is 5.

(Refer Slide Time: 39:33)



```
GettingStarted.ipynb
File Edit View Insert Runtime Tools Help
+ Code + Text
RAM 1.0 GB
Disk 10.0 GB
editing

[35] 1 print(nx.diameter(G))
5

Density of a network

$$D(G) = \frac{E}{E_{max}}$$


[36] 1 print(nx.density(G))
0.13983743315588021

Connected components

[37] 1 print(nx.number_connected_components(G))
1

Clique

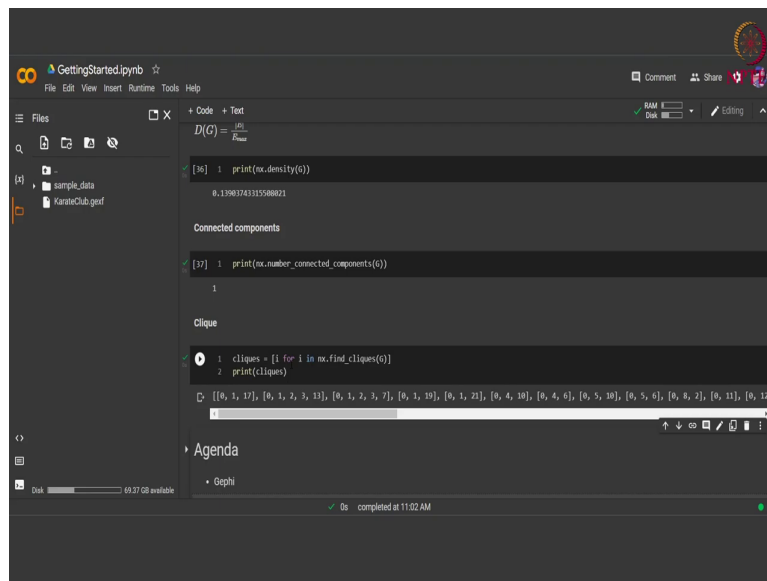
[ ] 1 cliques = [i for i in nx.find_cliques(G)]
2 print(cliques)
```

Then again we can also calculate the density of the graph, that is the number of actual edges that are present divided by the number of all the possible edges that there can be in the graph in order to see the how dense the network is.

So, for the our particular karate club graph we are getting the density as 0.13 then the other thing that we can see are the number of connected components that is whether do we have any isolated nodes or isolated component graph basically two sub graphs in our network which are not connected by any edge.

So, we see here and we get a connected component of 1 that is each node is reachable by all the other nodes.

(Refer Slide Time: 40:17)



The screenshot shows a Jupyter Notebook titled 'GettingStarted.ipynb'. The code cell contains the following Python code:

```
D(G) =  $\frac{E(G)}{|G|}$   
  
[36]: 1 print(nx.density(G))  
      2  
      3 0.15983743315588821  
  
Connected components  
  
[37]: 1 print(nx.number_connected_components(G))  
      2  
      3 1  
  
Clique  
  
[38]: 1 cliques = [i for i in nx.find_cliques(G)]  
      2 print(cliques)  
      3  
      4 [[0, 1, 17], [0, 1, 2, 3, 13], [0, 1, 2, 3, 7], [0, 1, 10], [0, 1, 21], [0, 4, 10], [0, 4, 6], [0, 5, 10], [0, 5, 6], [0, 6, 2], [0, 11], [0, 12]]
```

The output shows the density of the graph, the number of connected components, and a list of cliques. The cliques are: $[[0, 1, 17], [0, 1, 2, 3, 13], [0, 1, 2, 3, 7], [0, 1, 10], [0, 1, 21], [0, 4, 10], [0, 4, 6], [0, 5, 10], [0, 5, 6], [0, 6, 2], [0, 11], [0, 12]]$. The interface also shows a file browser on the left with 'sample_data' and 'KarateClub.gexf', and an agenda section with 'Gephi'.

And finally, we can calculate the cliques that are a sub graph which is basically like they all the nodes in the subgraph are connected with each other. So, we discussed a lot of network measures from NetworkX, but there are a number of more attributes that can be explored in the NetworkX library and I would leave it to you to explore all the other components like in the further classes.

Thank you.