Project 1

CSCI 6444: INTRODUCTION TO BIG DATA & ANALYTICS

Computer Science, George Washington University

Professor: Stephen Kaisler

Group 8

Shubhendra Singh   Vaishnavi Goyal

G31718255   G47669343

1. Dataset Description

The Epinions dataset is trust network dataset.from Who-trusts-whom network of Epinions.com..Names have been removed and replaced by numbers.The format for this dataset is <#FromNodeId> , <#ToNodeID >,<#Edges>. 75879 nodes that signifies the number of users of the Epinions Social Network and 508837 edges give information about the relationship between them. This dataset contains data in the form of a directed graph from one node(user) to another node(user).

2. Install the igraph package from one of the CRAN mirrors and loading the library in Rstudio

This is code to install the igraph package and load this library which was developed by Gábor Csárdi and Tamás Nepusz will act as an aid for creating, manipulating and visualizing of graphs.As in this project, we are going to use this package for social network analysis.

```
> install.packages("igraph")
WARNING: Rtools is required to build R packages but is not currently installed. Please download and install the appropriate version of Rtools before proc
eeding:

https://cran.rstudio.com/bin/windows/Rtools/
Installing package into 'C:/Users/ADMIN/AppData/Local/R/win-library/4.3'
(as 'lib' is unspecified)
trying URL 'https://cran.rstudio.com/bin/windows/contrib/4.3/igraph_2.0.2.zip'
Content type 'application/zip' length 7249034 bytes (6.9 MB)
downloaded 6.9 MB

package 'igraph' successfully unpacked and MD5 sums checked
Warning in install.packages :
  cannot remove prior installation of package 'igraph'
Warning in install.packages :
  problem copying C:\Users\ADMIN\AppData\Local\R\win-library\4.3\00LOCK\igraph\libs\x64\igraph.dll to C:\Users\ADMIN\AppData\Local\R\win-library\4.3\igra
ph\libs\x64\igraph.dll: Permission denied
Warning in install.packages :
  restored 'igraph'

The downloaded binary packages are in
        C:\Users\ADMIN\AppData\Local\Temp\RtmpIn9lWN\downloaded_packages
> library(igraph)

Attaching package: 'igraph'

The following objects are masked from 'package:stats':

    decompose, spectrum

The following object is masked from 'package:base':

    union

>
```

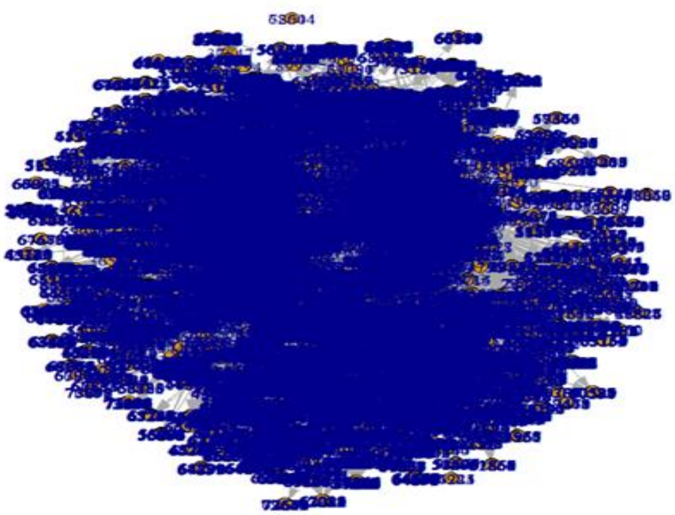
Import the specified dataset

By keeping the dataset in the working directory ,we load the dataset soc-Epinions1_adj.tsv file.For further analysis, it is converted into matrix named as optab and extract into vectors v1 and v2 to store two vertices .As the third column has information about edges which stores only one for all rows so we store in vector v3.Then we create relations using data frame between two vectors v1 and v2 and finally create a graph using graph_from_data_frame.

Code Snippet

```
> opinions<-read.table("soc-Epinions1_adj.tsv")
> optab<-as.matrix(opinions)
> v1<-optab[,1]
> v2<-optab[,2]
> v3<-optab[,3]
> relations<-data.frame(from=v1,to=v2)
> g<-graph_from_data_frame(relations,directed=TRUE)
> plot(g,vertex.size=5,edge.arrow.size=0.1)
> |
```

Once we plot a graph we see a huge bluebob shape structure which contains 75k edges . This graph looks incomprehensible due to vertex labels. Plotting of this graph required high execution time. To comprehend the graph we reduced the edge size to 0.1 and vertices size to 5 but it could not help in understanding the graph.



b. Determine how to create a graph and plot. Show the plot in your report.

As the structure shown above is intricate and could not provide us with the information that is required for analysis so we will be plotting definite portions of the graph as shown in code snippet below. We plotted the first 100 nodes,200 nodes , 500 nodes, 10000 nodes ,10000 nodes .

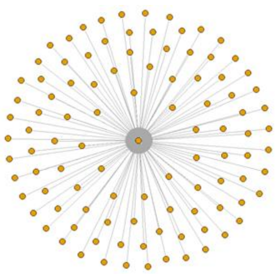
```
> first_100_vertices <- graph_from_data_frame(relations[1:100,])
> plot(first_100_vertices,vertex.size=5,edge.arrow.size=.4,vertex.label=NA)
> |
```

```
> first_200_vertices <- graph_from_data_frame(relations[1:200,])
> plot(first_200_vertices,vertex.size=4,edge.arrow.size=.4,vertex.label=NA)
> |
```

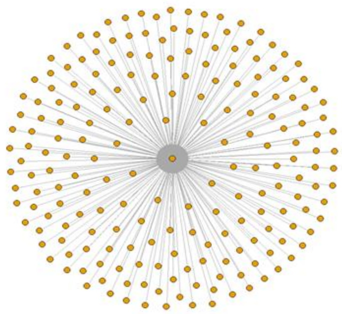
```
> first_500_vertices <- graph_from_data_frame(relations[1:500,])
> plot(first_500_vertices,vertex.size=5,edge.arrow.size=.4,vertex.label=NA)
>
```

```
> first_1000_vertices <- graph_from_data_frame(relations[1:1000,])
> plot(first_1000_vertices,vertex.size=5,edge.arrow.size=.4,vertex.label=NA)
>
```

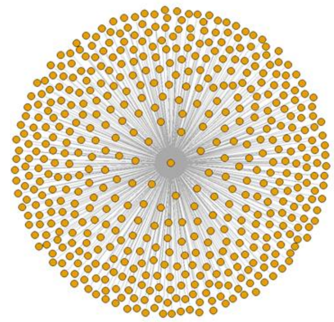
```
> first_10000_vertices <- graph_from_data_frame(relations[1:10000,])
> plot(first_10000_vertices,vertex.size=5,edge.arrow.size=.4,vertex.label=NA)
```



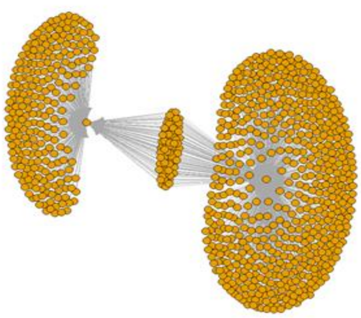
First 100 Vertices



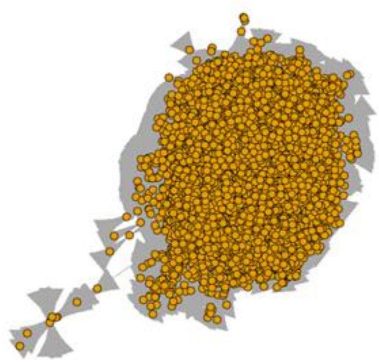
First 200 Vertices



First 500 Vertices



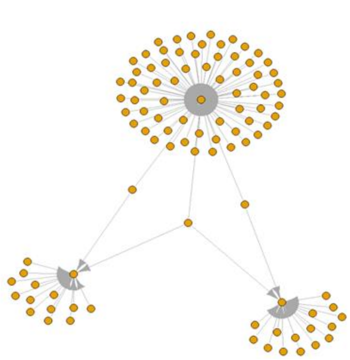
First 1000 Vertices



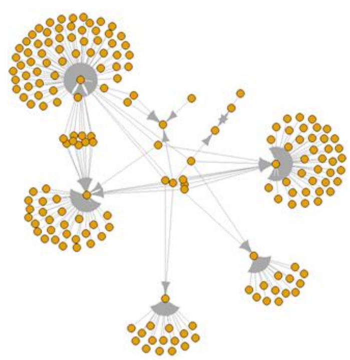
First 10000 Vertices

We try to summarize by plotting the last 100 nodes , 200 nodes , 500 nodes , 1000 nodes , 10000 nodes and the code snippet for the same is shown below.

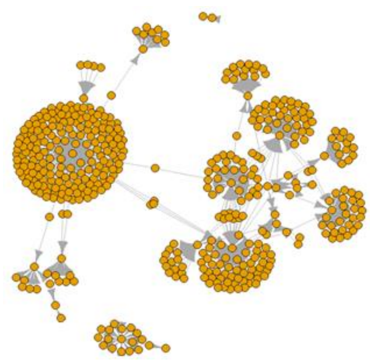
```
> last_100_g <- graph_from_data_frame(relations[(nrow(relations) -99):nrow(relations), ])
> plot(last_100_g,vertex.size=5,edge.arrow.size=.4,vertex.label=NA)
>

> last_200_g <- graph_from_data_frame(relations[(nrow(relations) - 199):nrow(relations), ])
> plot(last_200_g,vertex.size=5,edge.arrow.size=.4,vertex.label=NA)
>

> last_500_g <- graph_from_data_frame(relations[(nrow(relations) -499):nrow(relations), ])
> plot(last_500_g,vertex.size=5,edge.arrow.size=.4,vertex.label=NA)

> last_1000_g <- graph_from_data_frame(relations[(nrow(relations) -999):nrow(relations), ])
> plot(last_1000_g,vertex.size=5,edge.arrow.size=.4,vertex.label=NA)
>

> last_10000_g <- graph_from_data_frame(relations[(nrow(relations) -9999):nrow(relations), ])
> plot(last_10000_g,vertex.size=5,edge.arrow.size=.4,layout=layout, vertex.label=NA)
> layout <- layout_with_fr(last_10000_g, niter=800, area=25, repulserad=1.5)
Warning messages:
1: In layout_with_fr(last_10000_g, niter = 800, area = 25, repulserad = 1.5) :
  Argument `area' is deprecated and has no effect
2: In layout_with_fr(last_10000_g, niter = 800, area = 25, repulserad = 1.5) :
  Argument `repulserad' is deprecated and has no effect
>
> plot(last_10000_g,vertex.size=5,edge.arrow.size=.4,layout=layout, vertex.label=NA)
>
```



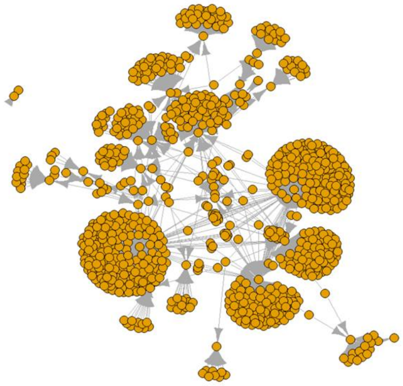
Last 100 vertices

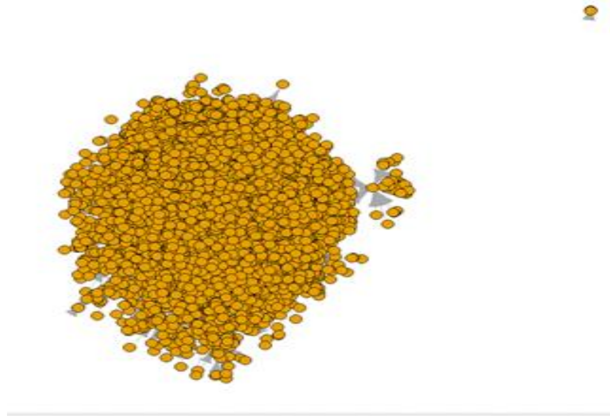


Last 200 Vertices



Last 500 Vertices





Last 1000 Vertices　　　　Last 10000 Vertices

3.Applying the functions shown in the Introduction to Graph Analytics document on Blackboard on the graph generated from the data set.

a. V(g) -Vertices of a graph

```
> V(g)
+ 75879/75879 vertices, named, from 8efb872:
   [1] 3     4     115   150   182   226   282   337   371   448   559   670   780   826   875   891   897   925   1002  1111  1112
  [22] 1122  1223  1334  1445  1556  1667  1778  1834  1889  2000  2001  2080  2111  2149  2222  2223  2242  2334  2346  2434  2445
  [43] 2501  2534  2556  2601  2623  2667  2727  2778  2811  2889  3000  3041  3089  3110  3111  3145  3222  3305  3333  3334  3379
  [64] 3410  3445  3534  3556  3574  3667  3778  3889  3989  4000  4111  4158  4222  4333  4341  4444  4445  4556  4563  4667  4778
  [85] 4889  4978  5000  5111  5222  5289  5333  5367  5385  5444  5456  5489  5554  5555  5556  5633  5666  5667  5744  5766  5777
 [106] 5778  5804  5888  5911  5922  5999  6000  6110  6155  6221  6244  6266  6332  6346  6355  6443  6554  6665  6666  6711  6777
 [127] 6789  6855  6888  6922  6999  7110  7221  7266  7332  7443  7444  7554  7665  7776  7777  7800  7888  7965  7998  8109  8220
 [148] 8331  8442  8553  8664  8691  8705  8775  8789  8886  8887  8998  9010  9072  9109  9220  9331  9442  9553  9646  9664  9775
 [169] 9831  9885  9886  9985  9997  9998  10109 10175 10220 10236 10331 10442 10553 10664 10720 10775 10886 10997 11109 11110 11221
 [190] 11332 11443 11479 11488 11554 11555 11578 11588 11665 11688 11776 11777 11866 11887 11943 11998 12109 12116 12187 12220 12221
+ ... omitted several vertices
```

The V(g) function retrieves the number of nodes in the graph as shown in the first line it fetches 75879 nodes which represents the total number of users in the network .For the sake of privacy , users names are replaced by numbers.

b. E(g): Edges of a Graph

```
> E(g)
+ 811480/811480 edges from 8efb872 (vertex names):
  [1] 3    ->1 4    ->1 115 ->1 150 ->1 182 ->1 226 ->1 282 ->1 337 ->1 371 ->1 448 ->1 559 ->1 670 ->1 780 ->1 826 ->1 875 ->1 891 ->1
 [17] 897 ->1 925 ->1 1002->1 1111->1 1112->1 1122->1 1223->1 1334->1 1445->1 1556->1 1667->1 1778->1 1834->1 1889->1 2000->1 2001->1
 [33] 2080->1 2111->1 2149->1 2222->1 2223->1 2242->1 2334->1 2346->1 2434->1 2445->1 2501->1 2534->1 2556->1 2601->1 2623->1 2667->1
 [49] 2727->1 2778->1 2811->1 2889->1 3000->1 3041->1 3089->1 3110->1 3111->1 3145->1 3222->1 3305->1 3333->1 3334->1 3379->1 3410->1
 [65] 3445->1 3534->1 3556->1 3574->1 3667->1 3778->1 3889->1 3989->1 4000->1 4111->1 4158->1 4222->1 4333->1 4341->1 4444->1 4445->1
 [81] 4556->1 4563->1 4667->1 4778->1 4889->1 4978->1 5000->1 5111->1 5222->1 5289->1 5333->1 5367->1 5385->1 5444->1 5456->1 5489->1
 [97] 5554->1 5555->1 5556->1 5633->1 5666->1 5667->1 5744->1 5766->1 5777->1 5778->1 5804->1 5888->1 5911->1 5922->1 5999->1 6000->1
[113] 6110->1 6155->1 6221->1 6244->1 6266->1 6332->1 6346->1 6355->1 6443->1 6554->1 6665->1 6666->1 6711->1 6777->1 6789->1 6855->1
[129] 6888->1 6922->1 6999->1 7110->1 7221->1 7266->1 7332->1 7443->1 7444->1 7554->1 7665->1 7776->1 7777->1 7800->1 7888->1 7965->1
[145] 7998->1 8109->1 8220->1 8331->1 8442->1 8553->1 8664->1 8691->1 8705->1 8775->1 8789->1 8886->1 8887->1 8998->1 9010->1 9072->1
+ ... omitted several edges
```

The E(g) function retrieves the number of edges of the graph which is 811480. This information will definitely help in analyzing the network between the vertices.

c. get adjacency(): Adjacency of graph

```
> Adjacency_graph<-igraph::get.adjacency(g)
Warning message:
`get.adjacency()` was deprecated in igraph 2.0.0.
ℹ Please use `as_adjacency_matrix()` instead.
This warning is displayed once every 8 hours.
Call `lifecycle::last_lifecycle_warnings()` to see where this warning was generated.
> Adjacency_graph
```

```
> Adjacency_graph
75879 x 75879 sparse Matrix of class "dgCMatrix"
  [[ suppressing 60 column names ‘3’, ‘4’, ‘115’ ... ]]
  [[ suppressing 60 column names ‘3’, ‘4’, ‘115’ ... ]]

3   . 1 . . 1 1 . 1 . 1 . . . . . . . . . 1 1 . . . . . . . . 1 . 1 . 1 . 1 . . . 1 1 . 1 1 1 1 1 1 . . . . . . . . . . . . ......
4   1 . . . . 1 . 1 . 1 . 1 1 . . 1 . . 1 1 . . . . . . . 1 . 1 . . . 1 . 1 1 . . 1 1 1 . 1 1 . 1 1 . . . . . . . . . . . . ......
115 . . . . . . . 1 . 1 1 . . . . 1 . . . 1 . . . . 1 . . . . 1 . . . 1 . 1 . . . . . 1 . . . . . . . . . . 1 . . . . . . 1 ......
150 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . ......
182 1 . . . . . . 1 1 1 . . . . . 1 . . . . . . 1 1 1 . . . . . . . . . . . . . . . . . . . 1 . . . . . . . . . . . 1 . . . ......
226 1 1 . . . . . 1 1 1 . . 1 . . 1 . . 1 1 . . . . . . . 1 . 1 . 1 . 1 . 1 1 . . . . 1 . . 1 1 . 1 . 1 . . 1 . . . . . . . ......
282 . . . . . . . . . . . . . . . . . . . . 1 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . ......
337 1 1 1 . 1 1 . . . 1 . . 1 . . 1 . . 1 1 1 . . . . . . 1 1 1 . . . 1 . 1 1 . . . 1 1 . . 1 . . . . 1 . . . . . . . 1 . . ......

 ..............................
 ........suppressing 75819 columns and 75863 rows in show(); maybe adjust 'options(max.print= *, width = *)'
 ..............................
  [[ suppressing 60 column names ‘3’, ‘4’, ‘115’ ... ]]

41782 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . ......
41783 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . ......
41784 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . ......
41785 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . ......
41786 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . ......
41789 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . ......
41790 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . ......
41792 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . ......
```

The get.adjaceny () function is useful to fetch the adjacency matrix of a graph .As there are 75879 nodes in the graph which will correspond into 75879*75879 size of adjacency matrix.With the help of the adjacency matrix we can find out the nodes which are isolated as in the above case we can see the 150 ,282 vertices have less connectivity .It can help in further analysis of the graph.

d. gden(g) : Density of Graph

```
> Desity_Graph<-gden(relations)
> Desity_Graph
[1] 0.1569413
>
```

The gden() function is useful to calculate the graph density of the graph .A graph with higher density is more connected and can resist failure in the network .For the above graph ,we have a graph density as 0.15369413 which has less edges as compared to the all the possible edges. Therefore we can infer that our graph is sparse.

e. edge_density(g): Edge Density

```
> Edge_Density<-igraph::edge_density(g)
> Edge_Density
[1] 0.000140942
> igraph::edge_density(g,loops=TRUE)
[1] 0.0001409401
> 
```

The edge_density(g) function is used for determining edge density of a graph. Edge density is the ratio of the number of edges present in a graph to the maximum number of edges that might exist in the graph. This is evident from the edge density as well that our graph is sparse. Furthermore , when we have considered the loops as well it comes out to be the same which shows that there are no loops and we can infer that it is a simple graph.

f. degree(): Graph Degree

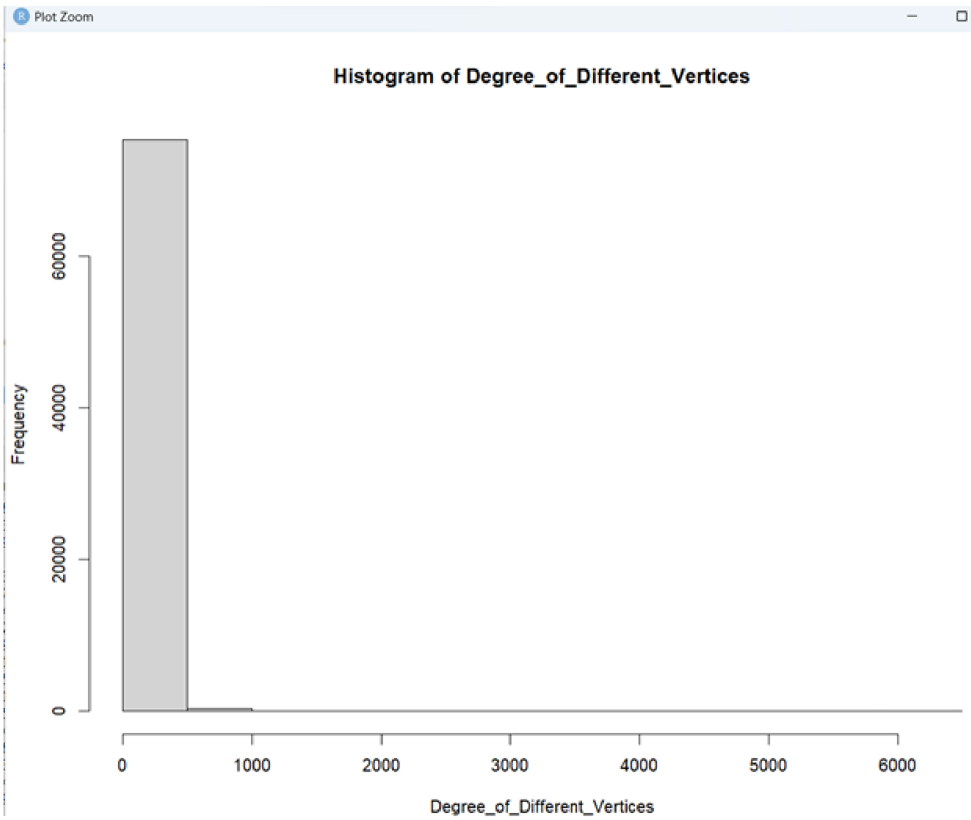
```
> Degree_of_Different_Vertices<-igraph::degree(g)
> Degree_of_Different_Vertices
    3     4   115   150   182   226   282   337   371   448   559   670   780   826   875   891   897   925  1002  1111  1112  1122
  572   690   686    48   636   880   266   924   342   826   618   226   828    38    14   324    40   372   332   448   498    22
 1223  1334  1445  1556  1667  1778  1834  1889  2000  2001  2080  2111  2149  2222  2223  2242  2334  2346  2434  2445  2501  2534
  362   830   508    82   366   432   216   524  2014   210    30   832    26   834   542    30    24   176   252   422  1046   224
 2556  2601  2623  2667  2727  2778  2811  2889  3000  3041  3089  3110  3111  3145  3222  3305  3333  3334  3379  3410  3445  3534
  602   352   170   510    38  1366    22   396   730    36   314    36  1736   814   104    20   732   820   666    12   144   206
 3556  3574  3667  3778  3889  3989  4000  4111  4158  4222  4333  4341  4444  4445  4556  4563  4667  4778  4889  4978  5000  5111
  362    28   390   262  1296    96  2410   236     8   664   364    20   412   640   980    22   332  3198    90   362  1328    86
 5222  5289  5333  5367  5385  5444  5456  5489  5554  5555  5556  5633  5666  5667  5744  5766  5777  5778  5804  5888  5911  5922
   80   760    36   602    66    52    60   588   112    46   128   310    10   398   352   370     8   432    24    56   522   764
 5999  6000  6110  6155  6221  6244  6266  6332  6346  6355  6443  6554  6665  6666  6711  6777  6789  6855  6888  6922  6999  7110
  612   650   288   232   186   240   226    16    20   650    96   208   222    10    96   140   110   962    16  1134   116    92
 7221  7266  7332  7443  7444  7554  7665  7776  7777  7800  7888  7965  7998  8109  8220  8331  8442  8553  8664  8691  8705  8775
  684   174   108   340   212   142   890   268     6   350     8   192   460    10    22   150    24    24   170    12    40     2
 8789  8886  8887  8998  9010  9072  9109  9220  9331  9442  9553  9646  9664  9775  9831  9885  9886  9985  9997  9998 10109 10175
  120  6088   132    28   158    32     6    74    14   234    84    24   714   126   488    18    60   130   976    42   302    54
10220 10236 10331 10442 10553 10664 10720 10775 10886 10997 11109 11110 11221 11332 11443 11479 11488 11554 11555 11578 11588 11665
  154     8   272    86   754     8    36   108   444    90   466   104   382    48   364    18   428     2   390    10   360    34
11688 11776 11777 11866 11887 11943 11998 12109 12116 12187 12220 12221 12321 12332 12388 12443 12552 12554 12625 12665 12776 12788
  512    70   362    64    66   326   112    14    24   172   530   810   254    78   666   494    12    88    42    60   444   570
12799 12887 12998 13065 13109 13143 13220 13276 13309 13331 13332 13443 13448 13554 13643 13665 13688 13776 13887 13954 13998 14109
  348   224   606   386     4   432    40   300   292   682     8    32    12   364   710   608   420    28   560   150    72     6
14110 14154 14176 14187 14198 14209 14220 14276 14298 14331 14442 14443 14554 14665 14710 14776 14887 14998 15109 15220 15331 15442
  184   192   212   518    44   694   190   416   318    72   158    14    30   130   158   552   236   588   396   320   504   268
15553 15554 15665 15765 15776 15810 15832 15887 15921 15998 16109 16220 16331 16420 16442 16553 16631 16664 16665 16666 16765 16776
  514    34   156   192   134   156   170   104   240    76   412   112    30   388   168   290   528   568   148   114   426   122
16821 16887 16998 17010 17109 17220 17331 17365 17442 17466 17532 17553 17560 17664 17775 17776 17887 17950 17998 18109 18110 18220
  264    84     6    40   220    58    12   420   158     8    12     2    20    12   618    76   138    10    78     6   340    28
18287 18331 18343 18442 18553 18609 18664 18665 18775 18831 18867 18886 18887 18998 19109 19166 19198 19220 19287 19331 19409 19442
  506    10   396    24    36   122    14   332   140   232     4  2032   120     2    30    12   198    34   192    30   168     2
19443 19454 19460 19542 19553 19664 19754 19775 19886 19997 19998 20109 20220 20227 20331 20442 20553 20664 20742 20775 20886 20997
  302   164     8    80    56   230     6     4   786  1572    74   104   560    28   168    32   100   204   528   242   122   632
21108 21109 21179 21220 21331 21411 21442 21476 21553 21653 21664 21720 21775 21776 21886 21997 22042 22108 22164 22220 22221 22250
  740   198    26   506    86    14    92   182   474   148   890   116   318   298   446    28   336   632   594   908   274    14
22332 22443 22554 22632 22665 22776 22781 22887 22998 23032 23065 23109 23220 23320 23331 23366 23599 23639 23788 23865 23943 23998
  924   200   292   134    10    36    12   170   514   206   194    80    32    64  1186   154   156     8    72   102   280    82
24221 24442 24610 24732 24821 24864 24899 25553 25621 25987 26398 26409 26431 26664 26887 26910 27010 27031 27265 27309 27387 27587
  190   280    76   248   610     8    74   556   322    22    80   116   160  1740   422   110   268     8   352   114    22   342
27676 27775 28220 28376 28443 28620 28720 28886 29220 29849 29997 30254 30864 31108 31443 31498 31575 31664 31709 32086 32219 32510
```

The degree() function which is in the igraph package and used to compute the number of edges connected to each vertex. It helps in finding the number of neighboring edges . For further analysis ,we plot a histogram to get an insight about edge density of a graph and overall composition of a graph

g. Histogram

To visualize the distribution of the degrees of nodes in the graph, we'll create a histogram.

```
> hist(Degree_of_Different_Vertices)
>
```



From the above histogram,we can infer that the majority of the nodes are between the range of 0 to 500 and there are some nodes between 500 to 1000 .Furthermore , there are no edges between 1000 to 6000.

h. centr_betw()- Betweenness Centrality

The function centr_betw() is used to find betweenness centrality, which will be an aid for determining the number of shortest paths that pass through each vertex.

For instance, g has lower betweenness which suggests that there are a smaller number of nodes with the nodes that can significantly impact the network connectedness as we already inferred when computing edge density. This is useful in some situations, but it might also mean that some nodes in the graph are farther away than others, which isn't always a desirable thing.

```
> Graph_betweeness_Centrality<-igraph::centr_betw(g)
> Graph_betweeness_Centrality
$res
   [1] 3.151719e+06 3.981881e+06 3.266552e+06 1.206334e+05 7.363342e+06 9.469315e+06 1.701157e+06 8.122346e+06 2.725038e+06 5.273894e+06
  [11] 3.744818e+06 4.063249e+05 8.692060e+06 2.335483e+06 8.783339e+02 1.474652e+06 3.244700e+05 3.461340e+06 2.436035e+06 1.039556e+06
  [21] 7.101634e+06 9.004451e+05 2.007443e+06 1.553106e+07 2.858003e+06 1.403503e+05 4.735297e+06 1.712402e+06 2.053627e+06 2.216830e+06
  [31] 5.637427e+07 2.877706e+06 1.921803e+05 1.090843e+07 2.662356e+03 5.445164e+06 4.715068e+06 4.811867e+05 1.129239e+04 5.629457e+05
  [41] 9.515164e+05 2.272398e+06 2.380167e+07 1.479723e+06 2.850635e+06 4.453057e+06 9.638072e+05 3.046881e+06 1.930011e+05 2.534605e+07
  [51] 3.859493e+04 2.311553e+06 6.605852e+06 7.532751e+05 6.104356e+06 5.109320e+05 4.907386e+07 1.259179e+07 3.950718e+05 2.408225e+03
  [61] 1.068249e+07 1.672638e+07 7.387049e+06 1.527451e+05 1.285051e+06 1.412434e+06 2.400280e+06 1.313003e+04 3.802958e+06 4.471660e+06
  [71] 2.173360e+07 3.344439e+05 8.976368e+07 1.215126e+06 1.577523e+04 1.594366e+07 2.546772e+06 1.589372e+05 4.050165e+06 5.511125e+06
  [81] 1.211066e+07 1.410818e+04 3.086957e+06 1.609299e+08 8.555053e+05 2.563614e+06 2.308962e+07 4.421859e+05 1.343681e+06 8.883725e+06
  [91] 7.132626e+03 9.118853e+06 9.727240e+05 3.628466e+03 2.363156e+06 8.622036e+06 6.774917e+05 1.973869e+05 9.096726e+05 4.077454e+06
 [101] 4.890626e+02 5.032214e+06 1.153062e+07 2.657635e+06 3.215569e+01 3.119489e+06 2.223556e+05 2.428348e+04 3.576354e+06 2.992492e+07
 [111] 6.872374e+06 5.773731e+06 6.494667e+05 1.308536e+06 4.799707e+05 2.080087e+06 1.582126e+06 1.539632e+05 1.296378e+03 7.188281e+06
 [121] 1.513014e+05 1.261128e+05 1.018983e+06 1.520621e+05 1.951136e+05 2.601437e+06 3.520919e+05 2.423314e+07 2.710753e+02 2.146971e+07
 [131] 1.378418e+06 1.856139e+05 2.377946e+06 2.163510e+06 6.646553e+04 1.181188e+06 1.112392e+06 6.558604e+04 9.329885e+06 4.793107e+06
 [141] 8.509594e+01 2.131942e+06 4.316926e+01 1.355627e+06 7.031126e+06 4.947358e+02 1.570589e+04 3.756512e+05 7.708826e+04 2.088227e+05
 [151] 1.311206e+06 1.113333e+04 6.792458e+05 0.000000e+00 2.129580e+06 4.158361e+08 3.889832e+05 2.702684e+03 9.065062e+05 3.579090e+04
 [161] 5.335826e+02 1.104778e+04 2.928909e+03 1.278429e+06 1.269762e+05 1.079892e+06 6.699146e+06 3.022978e+05 1.189296e+07 1.847253e+05
 [171] 6.603394e+04 1.290064e+06 8.793124e+06 3.321546e+04 3.081221e+06 8.874598e+05 6.761301e+05 5.275565e+02 6.794543e+05 3.522564e+05
 [181] 6.649403e+06 1.520800e+05 1.985282e+04 2.180324e+05 3.251087e+06 1.178741e+05 4.053752e+06 1.462464e+05 6.780008e+06 4.608149e+05
```

```
 [991] 4.862595e+05 1.667332e+06 4.150323e+05 8.909589e+06 1.117529e+06 1.650705e+07 1.983223e+05 1.918185e+05 6.836492e+07 1.724489e+05
 [ reached getOption("max.print") -- omitted 74879 entries ]

$centralization
[1] 0.0721838

$theoretical_max
[1] 4.368596e+14
```

i. centr_clog (): Closeness Centrality

```
> Graph_closeness_Centrality<-igraph::centr_clo(g)
> Graph_closeness_Centrality
$res
   [1] 0.3235830 0.3182144 0.3204927 0.2823251 0.3207840 0.3293043 0.3087729 0.3229494 0.3021660 0.3246296 0.3207826 0.2975331 0.3315679
  [14] 0.2943417 0.2642594 0.3140524 0.2716610 0.3160723 0.3123369 0.3177786 0.3239823 0.2667126 0.3209699 0.3424100 0.3161158 0.2923683
  [27] 0.3077684 0.3178638 0.3058456 0.3158092 0.3518479 0.3082924 0.2727371 0.3334139 0.2699349 0.3197984 0.3230250 0.2749230 0.2786445
  [40] 0.3031511 0.3050439 0.3111585 0.3431999 0.3161987 0.3207338 0.3198981 0.3099168 0.3268967 0.2949711 0.3353028 0.2754350 0.3199953
  [53] 0.3284334 0.2871730 0.3058234 0.2739839 0.3554810 0.3386626 0.2808288 0.2706291 0.3264200 0.3364954 0.3282544 0.2685268 0.3009913
  [66] 0.3027798 0.3189461 0.2903622 0.3216570 0.3056891 0.3388955 0.3063370 0.3585687 0.3120068 0.2611470 0.3222253 0.3155636 0.2713161
  [79] 0.3162277 0.3303021 0.3366984 0.2834219 0.3070088 0.3623253 0.2915404 0.3218726 0.3302819 0.3036437 0.3008827 0.3308768 0.2756131
  [92] 0.3384390 0.2847589 0.2789292 0.2934538 0.3306302 0.3060861 0.2779269 0.3065214 0.3219778 0.2691621 0.3270278 0.3197081 0.3177414
 [105] 0.2569664 0.3195344 0.2753540 0.2850606 0.3292143 0.3159775 0.3212213 0.3202370 0.3009663 0.3116672 0.3051727 0.3185003 0.3169278
 [118] 0.2670825 0.2597710 0.3247977 0.2874810 0.3021756 0.3127321 0.2596661 0.2942664 0.3012375 0.3016686 0.3420195 0.2708116 0.3397499
 [131] 0.2922264 0.2856067 0.3222595 0.3131374 0.2940862 0.3136331 0.3109749 0.2923367 0.3370020 0.3063927 0.2561172 0.3188201 0.2633769
 [144] 0.3013260 0.3235098 0.2732803 0.2679890 0.3040379 0.2842011 0.2759810 0.2950433 0.2595489 0.2664092 0.2532991 0.3036072 0.3821987
 [157] 0.2968301 0.2832959 0.3070821 0.2857745 0.2619955 0.2830032 0.2578413 0.3019556 0.3018607 0.2685401 0.3273367 0.3101955 0.3298053
 [170] 0.2701232 0.2839161 0.2977760 0.3348337 0.2848904 0.3188014 0.2930718 0.3036255 0.2754860 0.3055610 0.2885963 0.3281735 0.2568872
 [183] 0.2894373 0.2983720 0.3220721 0.2924686 0.3193125 0.2818825 0.3171861 0.2838141 0.3066849 0.2537930 0.3259684 0.2532991 0.3188965
 [196] 0.2698024 0.3196098 0.2767208 0.3286482 0.2995937 0.3233871 0.2843512 0.2768783 0.3213070 0.2970974 0.2659106 0.2659432 0.3185885
 [209] 0.3238385 0.3351473 0.3059738 0.2762563 0.3228903 0.3251513 0.2835755 0.2988303 0.2964196 0.3030446 0.3206268 0.3221418 0.3252168
 [222] 0.3049813 0.3314158 0.3219778 0.2569820 0.3182424 0.2831353 0.3076711 0.3148774 0.3109710 0.2604103 0.2753650 0.2692443 0.3130792
```

```
 [989] 0.3088433 0.2847995 0.2897225 0.3112453 0.2906280 0.3286895 0.2938527 0.3077734 0.2953293 0.2978929 0.3435402 0.2820784
 [ reached getOption("max.print") -- omitted 74879 entries ]

$centralization
[1] 0.7633521

$theoretical_max
[1] 75877

>
```

The centr_clog(g) function is used to return the vicinity and centrality of vertices .As we can see from the above graph that we have visualized ,they have high connectivity which is verified through our average closeness centrality value that is 0.3.As it suggested in rubric it states that higher the closeness centrality ,higher they are strongly connected which is evident in our graph g.

j. shortest.paths(): Shortest Path between two nodes

First 200 vertices

```
> first_shortest_PATH_200_vertices=igraph::shortest.paths(first_200_vertices)
> first_shortest_PATH_200_vertices
    3 4 115 150 182 226 282 337 371 448 559 670 780 826 875 891 897 925 1002 1111 1112 1122 1223 1334 1445 1556 1667 1778 1834
3   0 2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2    2    2    2    2    2    2    2    2    2    2    2
4   2 0   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2    2    2    2    2    2    2    2    2    2    2    2
115 2 2   0   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2    2    2    2    2    2    2    2    2    2    2    2
150 2 2   2   0   2   2   2   2   2   2   2   2   2   2   2   2   2   2    2    2    2    2    2    2    2    2    2    2    2
    1889 2000 2001 2080 2111 2149 2222 2223 2242 2334 2346 2434 2445 2501 2534 2556 2601 2623 2667 2727 2778 2811 2889 3000 3041
3      2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2
4      2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2
115    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2
150    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2
    3089 3110 3111 3145 3222 3305 3333 3334 3379 3410 3445 3534 3556 3574 3667 3778 3889 3989 4000 4111 4158 4222 4333 4341 4444
3      2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2
4      2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2
115    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2
150    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2
    4445 4556 4563 4667 4778 4889 4978 5000 5111 5222 5289 5333 5367 5385 5444 5456 5489 5554 5555 5556 5633 5666 5667 5744 5766
3      2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2
4      2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2
115    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2
150    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2
    5777 5778 5804 5888 5911 5922 5999 6000 6110 6155 6221 6244 6266 6332 6346 6355 6443 6554 6665 6666 6711 6777 6789 6855 6888
```

The shortest.path() function is used to find shortest path in the range of two vertices .For above snapshot we have chosen to find shortest path between 200 vertices and we can infer that most of the paths are of value 2 .

First shortest path 1000 vertices

```
> first_1000_vertices <- graph_from_data_frame(relations[1:1000,])
> first_shortest_PATH_1000_vertices=igraph::shortest.paths(first_1000_vertices)
> first_shortest_PATH_1000_vertices
  3 4 115 150 182 226 282 337 371 448 559 670 780 826 875 891 897 925 1002 1111 1112 1122 1223 1334 1445 1556 1667 1778 1834
3 0 2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2    2    2    2    2    2    2    2    2    2    2    2
  1889 2000 2001 2080 2111 2149 2222 2223 2242 2334 2346 2434 2445 2501 2534 2556 2601 2623 2667 2727 2778 2811 2889 3000 3041
3    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2
  3089 3110 3111 3145 3222 3305 3333 3334 3379 3410 3445 3534 3556 3574 3667 3778 3889 3989 4000 4111 4158 4222 4333 4341 4444
3    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2
  4445 4556 4563 4667 4778 4889 4978 5000 5111 5222 5289 5333 5367 5385 5444 5456 5489 5554 5555 5556 5633 5666 5667 5744 5766
3    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2
  5777 5778 5804 5888 5911 5922 5999 6000 6110 6155 6221 6244 6266 6332 6346 6355 6443 6554 6665 6666 6711 6777 6789 6855 6888
3    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2
  6922 6999 7110 7221 7266 7332 7443 7444 7554 7665 7776 7777 7800 7888 7965 7998 8109 8220 8331 8442 8553 8664 8691 8705 8775
3    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2
  8789 8886 8887 8998 9010 9072 9109 9220 9331 9442 9553 9646 9664 9775 9831 9885 9886 9985 9997 9998 10109 10175 10220 10236
3    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2    2     2     2     2     2
  10331 10442 10553 10664 10720 10775 10886 10997 11109 11110 11221 11332 11443 11479 11488 11554 11555 11578 11588 11665 11688
3     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2
  11776 11777 11866 11887 11943 11998 12109 12116 12187 12220 12221 12321 12332 12388 12443 12552 12554 12625 12665 12776 12788
3     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2
  12799 12887 12998 13065 13109 13143 13220 13276 13309 13331 13332 13443 13448 13554 13643 13665 13688 13776 13887 13954 13998
3     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2
  14109 14110 14154 14176 14187 14198 14209 14220 14276 14298 14331 14442 14443 14554 14665 14710 14776 14887 14998 15109 15220
3     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2
  15331 15442 15553 15554 15665 15765 15776 15810 15832 15887 15921 15998 16109 16220 16331 16420 16442 16553 16631 16664 16665
3     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2
  16666 16765 16776 16821 16887 16998 17010 17109 17220 17331 17365 17442 17466 17532 17553 17560 17664 17775 17776 17887 17950
3     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2
  17998 18109 18110 18220 18287 18331 18343 18442 18553 18609 18664 18665 18775 18831 18867 18886 18887 18998 19109 19166 19198
3     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2
  19220 19287 19331 19409 19442 19443 19454 19460 19542 19553 19664 19754 19775 19886 19997 19998 20109 20220 20227 20331 20442
3     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2
  20553 20664 20742 20775 20886 20997 21108 21109 21179 21220 21331 21411 21442 21476 21553 21653 21664 21720 21775 21776 21886
3     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2
  21997 22042 22108 22164 22220 22221 22250 22332 22443 22554 22632 22665 22776 22781 22887 22998 23032 23065 23109 23220 23320
3     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2
  23331 23366 23599 23639 23788 23865 23943 23998 24221 24442 24610 24732 24821 24864 24899 25553 25621 25987 26398 26409 26431
3     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2
  26664 26887 26910 27010 27031 27265 27309 27387 27587 27676 27775 28220 28376 28443 28620 28720 28886 29220 29849 29997 30254
```

We tried to analyze for another section of the graph by finding shortest path between 1000 vertices and get the same result that most of them have value of the paths are of 2

k. get.shortest.path(): Get shortest paths between vertices in a graph

```
[1] 282  1112 2    5237

$vpath[[755]]
+ 3/75879 vertices, named, from 1802042:
[1] 282  4778 5278

$vpath[[756]]
+ 3/75879 vertices, named, from 1802042:
[1] 282  3889 5433

$vpath[[757]]
+ 3/75879 vertices, named, from 1802042:
[1] 282  4778 5467

$vpath[[758]]
+ 4/75879 vertices, named, from 1802042:
[1] 282   1112  74214 5496

$vpath[[759]]
+ 3/75879 vertices, named, from 1802042:
[1] 282  8886 5507

$vpath[[760]]
```

The get.shortest.path function helps in computing the length of the shortest path from the given node that we describe in a function .For above case we have chosen the node 7 for our graph g to check the path to various nodes from node 7 and it comes out to be 3 in most of the cases which suggest it is strongly connected.

l. max_cliques(): Max Cliques

```
> node<-c(50)
> g_50clique=igraph::max_cliques(g_Adjaceny_Graph,min = NULL,max = NULL,subset = node)
> g_50clique
[[1]]
+ 2/75879 vertices, named, from 1ed9b8f:
[1] 43528 1324
```

Focusing on cliques, we explored groups of fully interconnected nodes within the graph. The max_cliques() function from the igraph package helped us identify the largest clique and all maximal cliques. Interestingly, for node 50, the largest clique comprised only nodes 43528 and 1324, indicating a strong connection between them, even if their connections to the broader network might be limited. Larger cliques would generally signify densely connected groups of nodes.

m. clique_num(): Largest clique in the graph

```
> Largest_cliques_g <-igraph::clique_num(g_Adjaceny_Graph)
Warning message:
In igraph::clique_num(g_Adjaceny_Graph) :
  At vendor/cigraph/src/cliques/maximal_cliques_template.h:219 : Edge directions are ignored for maximal clique calculation.
> Largest_cliques_g
[1] 23
>
```

The clique_num() function determines the size of the greatest clique in the graph, which in our case is 23.

n. Geodesic : First 1000 vertices of geodist

Within a network, the shortest path connecting any two nodes is called a geodesic. A node is considered to have high betweenness if it frequently lies on these geodesics between many other node pairs. This implies that the removal or failure of such a high-betweenness node would have a greater impact on the overall connectivity of the network.

```
> Adjacency_matrix<-as.matrix(get.adjacency(first_1000_vertices,sparse=FALSE))
Warning message:
`get.adjacency()` was deprecated in igraph 2.0.0.
i Please use `as_adjacency_matrix()` instead.
This warning is displayed once every 8 hours.
Call `lifecycle::last_lifecycle_warnings()` to see where this warning was generated.
>
> geodesic_distances<-geodist(Adjacency_matrix)
> geodesic_distances
$counts
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22]
[1,]    1    0    0    0    0    0    0    0    0     0     0     0     0     0     0     0     0     0     0     0     0     0
     [,23] [,24] [,25] [,26] [,27] [,28] [,29] [,30] [,31] [,32] [,33] [,34] [,35] [,36] [,37] [,38] [,39] [,40] [,41] [,42]
[1,]     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     [,43] [,44] [,45] [,46] [,47] [,48] [,49] [,50] [,51] [,52] [,53] [,54] [,55] [,56] [,57] [,58] [,59] [,60] [,61] [,62]
[1,]     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     [,63] [,64] [,65] [,66] [,67] [,68] [,69] [,70] [,71] [,72] [,73] [,74] [,75] [,76] [,77] [,78] [,79] [,80] [,81] [,82]
[1,]     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     [,83] [,84] [,85] [,86] [,87] [,88] [,89] [,90] [,91] [,92] [,93] [,94] [,95] [,96] [,97] [,98] [,99] [,100] [,101] [,102]
[1,]     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0      0      0      0
     [,103] [,104] [,105] [,106] [,107] [,108] [,109] [,110] [,111] [,112] [,113] [,114] [,115] [,116] [,117] [,118] [,119]
[1,]      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
     [,120] [,121] [,122] [,123] [,124] [,125] [,126] [,127] [,128] [,129] [,130] [,131] [,132] [,133] [,134] [,135] [,136]
[1,]      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
     [,137] [,138] [,139] [,140] [,141] [,142] [,143] [,144] [,145] [,146] [,147] [,148] [,149] [,150] [,151] [,152] [,153]
[1,]      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
     [,154] [,155] [,156] [,157] [,158] [,159] [,160] [,161] [,162] [,163] [,164] [,165] [,166] [,167] [,168] [,169] [,170]
[1,]      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
     [,171] [,172] [,173] [,174] [,175] [,176] [,177] [,178] [,179] [,180] [,181] [,182] [,183] [,184] [,185] [,186] [,187]
[1,]      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
     [,188] [,189] [,190] [,191] [,192] [,193] [,194] [,195] [,196] [,197] [,198] [,199] [,200] [,201] [,202] [,203] [,204]
[1,]      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
     [,205] [,206] [,207] [,208] [,209] [,210] [,211] [,212] [,213] [,214] [,215] [,216] [,217] [,218] [,219] [,220] [,221]
[1,]      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
     [,222] [,223] [,224] [,225] [,226] [,227] [,228] [,229] [,230] [,231] [,232] [,233] [,234] [,235] [,236] [,237] [,238]
[1,]      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
     [,239] [,240] [,241] [,242] [,243] [,244] [,245] [,246] [,247] [,248] [,249] [,250] [,251] [,252] [,253] [,254] [,255]
[1,]      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
     [,256] [,257] [,258] [,259] [,260] [,261] [,262] [,263] [,264] [,265] [,266] [,267] [,268] [,269] [,270] [,271] [,272]
[1,]      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0      0
     [,273] [,274] [,275] [,276] [,277] [,278] [,279] [,280] [,281] [,282] [,283] [,284] [,285] [,286] [,287] [,288] [,289]
```

o. Matrix multiplication

Finding the number of paths between two nodes in a network can be done by multiplying the **adjacency matrix** by itself. This essentially counts both direct and **indirect**

connections through one intermediate node, providing a total count in the corresponding cell of the resulting matrix.

```
> Adjacency_matrix<-as.matrix(get.adjacency(first_1000_vertices,sparse=FALSE))
>
> Multiplication_of_matrix=Adjacency_matrix%*%Adjacency_matrix
> Multiplication_of_matrix
  3 4 115 150 182 226 282 337 371 448 559 670 780 826 875 891 897 925 1002 1111 1112 1122 1223 1334 1445 1556 1667 1778 1834
3 0 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0    0    0    0    0    0    0    0    0    0    0    0
  1889 2000 2001 2080 2111 2149 2222 2223 2242 2334 2346 2434 2445 2501 2534 2556 2601 2623 2667 2727 2778 2811 2889 3000 3041
3    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
  3089 3110 3111 3145 3222 3305 3333 3334 3379 3410 3445 3534 3556 3574 3667 3778 3889 3989 4000 4111 4158 4222 4333 4341 4444
3    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
  4445 4556 4563 4667 4778 4889 4978 5000 5111 5222 5289 5333 5367 5385 5444 5456 5489 5554 5555 5556 5633 5666 5667 5744 5766
3    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
  5777 5778 5804 5888 5911 5922 5999 6000 6110 6155 6221 6244 6266 6332 6346 6355 6443 6554 6665 6666 6711 6777 6789 6855 6888
3    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
  6922 6999 7110 7221 7266 7332 7443 7444 7554 7665 7776 7777 7800 7888 7965 7998 8109 8220 8331 8442 8553 8664 8691 8705 8775
3    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
  8789 8886 8887 8998 9010 9072 9109 9220 9331 9442 9553 9646 9664 9775 9831 9885 9886 9985 9997 9998 10109 10175 10220 10236
3    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0     0     0     0     0
  10331 10442 10553 10664 10720 10775 10886 10997 11109 11110 11221 11332 11443 11479 11488 11554 11555 11578 11588 11665 11688
3     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
  11776 11777 11866 11887 11943 11998 12109 12116 12187 12220 12221 12321 12332 12388 12443 12552 12554 12625 12665 12776 12788
3     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
  12799 12887 12998 13065 13109 13143 13220 13276 13309 13331 13332 13443 13448 13554 13643 13665 13688 13776 13887 13954 13998
3     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
  14109 14110 14154 14176 14187 14198 14209 14220 14276 14298 14331 14442 14443 14554 14665 14710 14776 14887 14998 15109 15220
3     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
  15331 15442 15553 15554 15665 15765 15776 15810 15832 15887 15921 15998 16109 16220 16331 16420 16442 16553 16631 16664 16665
3     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
  16666 16765 16776 16821 16887 16998 17010 17109 17220 17331 17365 17442 17466 17532 17553 17560 17664 17775 17776 17887 17950
3     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
  17998 18109 18110 18220 18287 18331 18343 18442 18553 18609 18664 18665 18775 18831 18867 18886 18887 18998 19109 19166 19198
3     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
  19220 19287 19331 19409 19442 19443 19454 19460 19542 19553 19664 19754 19775 19886 19997 19998 20109 20220 20227 20331 20442
3     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
  20553 20664 20742 20775 20886 20997 21108 21109 21179 21220 21331 21411 21442 21476 21553 21653 21664 21720 21775 21776 21886
3     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
  21997 22042 22108 22164 22220 22221 22250 22332 22443 22554 22632 22665 22776 22781 22887 22998 23032 23065 23109 23220 23320
3     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
  23331 23366 23599 23639 23788 23865 23943 23998 24221 24442 24610 24732 24821 24864 24899 25553 25621 25987 26398 26409 26431
3     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
```

p. Simplify() and is.simple()

```
> is.simple(g)
[1] TRUE
> simplify_g<-simplify(g)
> is.simple(simplify_g)
[1] TRUE
> is.simple(g)
[1] TRUE
>
```

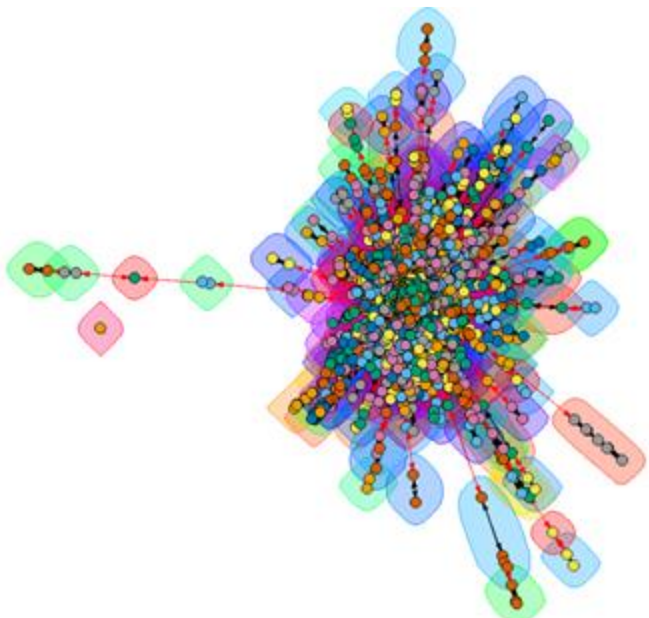
Our analysis confirms that the graph is **simple**, meaning it has no **loops** (edges connecting a node to itself) and **multiple edges** between any two nodes. This inherent simplicity is further validated by the is.simple() method consistently returning TRUE even after applying the simplify() function, which typically removes redundant edges.

q. Detecting Structures using walktrap.community()

This function uses random walks to search a network for highly connected subgraphs, commonly known as communities.

```
> wc<-walktrap.community(g)
Warning message:
`walktrap.community()` was deprecated in igraph 2.0.0.
ℹ Please use `cluster_walktrap()` instead.
This warning is displayed once every 8 hours.
Call `lifecycle::last_lifecycle_warnings()` to see where this warning was generated.
>
> plot(wc,g,vertex.size=5,edge.arrow.size=.4,vertex.label=NA,layout=layout.fruchterman.reingold)
```

In this we tried to analyze the communities in our network and as plotted below we can see densely connected communities and few isolated nodes.For informative visualization we have removed the labels of the nodes



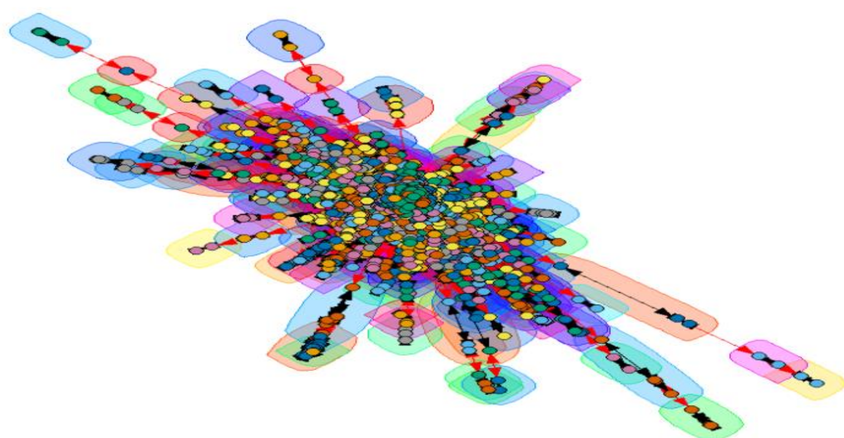
Graph Simplification

In this we step check if the above graph is simple and is simplify the graph by removing the loops and multiple edges of a graph .As our result suggest that our graph g is simple and is simplified through the simplify function.

```
> is.simple(g)
[1] TRUE
> simplify_g<-simplify(g)
> is.simple(simplify_g)
[1] TRUE
> is.simple(g)
[1] TRUE
>
```

Again we have plotted the graph and we can infer that our graph is simplified as there are no isolated edges and is strongly connected.

```
> wc<-walktrap.community(g)
Warning message:
`walktrap.community()` was deprecated in igraph 2.0.0.
i Please use `cluster_walktrap()` instead.
This warning is displayed once every 8 hours.
Call `lifecycle::last_lifecycle_warnings()` to see where this warning was generated.
>
> plot(wc,g,vertex.size=5,edge.arrow.size=.4,vertex.label=NA,layout=layout.fruchterman.reingold)
```



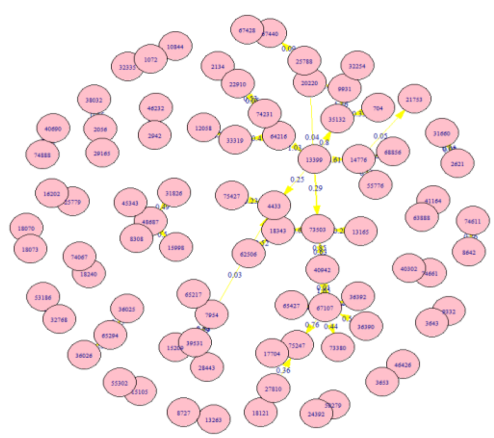
```
> sum(igraph::degree(g)==0)
[1] 0
```

To check if the graph g has any isolated vertices but from the result we can infer that there are no isolated vertices which verifies our graph is simplified.

```
> Graph_v2<-g
> E(Graph_v2)$weight<-rnorm(ecount(Graph_v2))
> V(Graph_v2)$weight<-rnorm(ecount(Graph_v2))
Error in i_set_vertex_attr(x, attr(value, "name"), index = value, value = attr(value,  :
  Length of new attribute value must be 1 or 75879, the number of target vertices, not 811480
> V(Graph_v2)$weight<-rnorm(vcount(Graph_v2))
> Subgraph_Duplicate<-induced.subgraph(Graph_v2,which(V(Graph_v2)$weight >2.2))
Warning message:
`induced.subgraph()` was deprecated in igraph 2.0.0.
i Please use `induced_subgraph()` instead.
This warning is displayed once every 8 hours.
Call `lifecycle::last_lifecycle_warnings()` to see where this warning was generated.
> plot(delete.vertices(Subgraph_Duplicate,degree(Subgraph_Duplicate)==0),edge.label=round(E(Subgraph_Duplicate)$weight,3),edge.arrow.size=0.1,
vertex.label.cex=0.5,edge.label.cex=0.3)
```

```
> plot(igraph::delete.vertices(Subgraph_Duplicate,igraph::degree(Subgraph_Duplicate)==0),edge.label=round(E(Subgraph_Duplicate)$weight,2),edge.arrow.size
=0.1,vertex.label.cex=0.5,edge.label.cex=0.6,vertex.color="Pink",edge.color="Yellow")
>
```

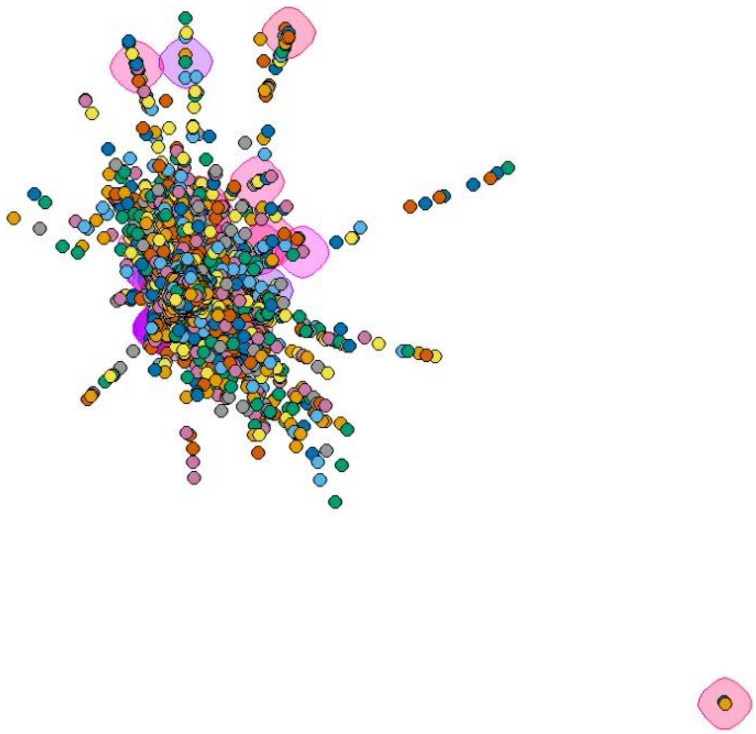
For further analysis of a graph,We copied the graph in Graph_v2 we assigned a random weight to a graph using rnorm function and then we induced a subgraph using induced.subgraph() function of choosing the vertices.We chose 2.2 weight as it was used in the rubric on the astrocollab dataset to induce a subgraph whose weights are greater than 2.2 . Furthermore, we have removed negative edges for further simplification. Then plotted the graph for the same.



Above subgraph visualizes the vertices that higher weighted nodes in the graph.

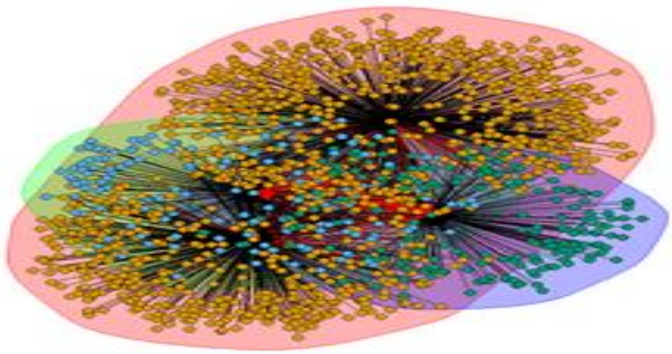
```
> plot(wc_subgraph_duplicate,g,vertex.size=4,vertex.label.cex=0.1,edge.arrow.size=0.1,layout=layout.fruchterman.reingold)
```

For further analysis , we plotted the walktrap to locate the communities for sub induced graph wc_subgraph_duplicate.



```
> first_2000_vertices <- graph_from_data_frame(relations[1:2000,])
> wc<-walktrap.community(first_2000_vertices)
Warning message:
`walktrap.community()` was deprecated in igraph 2.0.0.
ℹ Please use `cluster_walktrap()` instead.
This warning is displayed once every 8 hours.
Call `lifecycle::last_lifecycle_warnings()` to see where this warning was generated.
> plot(wc,first_2000_vertices,vertex.size=4,vertex.label.cex=0.1,edge.arrow.size=0.1,layout=layout.fruchterman.reingold)
>
> |
```

For analysis of graph we plotted the walktrap community graph for first 2000 nodes .This graph helps us to analyze through different walks that there are different communities which are strongly connected to each other.



First 2000 Vertices

4. Determining the alpha centrality, central node in the graph, longest paths, largest cliques, egos, and power centrality

a. Alpha Centrality

Our analysis identified node 50887 as the most influential node in the network, with a significantly higher alpha centrality score 40.748692 compared to others. Nodes 23710, 39753, and also has 16726 notable influence, but their scores 37.211, 33.050, and 32.650 are considerably lower than 50887.

```
> Alpha_Centrality_Subgraph_Duplicate=alpha.centrality(Subgraph_Duplicate)
Warning message:
`alpha.centrality()` was deprecated in igraph 2.0.0.
i Please use `alpha_centrality()` instead.
This warning is displayed once every 8 hours.
Call `lifecycle::last_lifecycle_warnings()` to see where this
warning was generated.
> sort(Alpha_Centrality_Subgraph_Duplicate,decreasing = TRUE)
    50887     23710     39753     16276      2285     23555 
40.748692 37.211820 33.050815 32.656582 31.280821 31.193787 
    11684     30043      7011      9606     10365      2613 
25.759832 21.658519 17.284832 11.981442 11.925774 10.495714 
     3174      7788     22999     28465     61328     51143 
10.287461  9.642084  9.123992  7.882656  7.499983  6.565762 
    31709     34172     40875      8985      5387     52442 
 6.548069  6.244471  5.257479  4.780928  4.180980  4.012484 
    13907      2523     68199     63684      9450     30595 
 3.653987  3.542953  3.541866  3.371682  3.349603  3.257399 
    13943     17832      9616      7463      6533     16196 
 3.161716  3.134067  3.120917  3.120008  2.360410  2.292783 
    47226     47839     75603     29698     22825      6714 
 2.268341  1.896203  1.840710  1.597750  1.570395  1.334687 
     1894     22051     22221     52785     48343     15589 
 1.323687  1.186475  1.114027  1.104698  1.059742  1.052177 
    30937     32747     75073     14467      9010     20227 
 1.031566  1.026132  1.018728  1.002675  1.000000  1.000000 
    27387     64839     62039     63773      1820     10534 
 1.000000  1.000000  1.000000  1.000000  1.000000  1.000000 
    17423     71049      4460     41906     31355     41915 
 1.000000  1.000000  1.000000  1.000000  1.000000  1.000000 
    41955     41976     30076      3335     21243      3055 
 1.000000  1.000000  1.000000  1.000000  1.000000  1.000000 
    15420     11141     11274     11533     42039      7706 
```

b. Central Node

The node with the highest betweenness can be used to determine the center node, or we can use the total of the in and out degrees. The central node 8886 is located as indicated below, based on the total of the in and out degrees.

```
> Betweenness_value<-igraph::betweenness(g)
> Central_Most_Value<-which.max(Betweenness_value)
> Central_Most_Value
8886 
 156 
> 
```

We discovered that the center node was the same 8886 based on the maximum betweenness value, as indicated below.

```
> Central_Node<-which.max(igraph::degree(g,mode="all"))
> Central_Node
8886
 156
```

c . Longest path(subgraph) : Longest path of the graph

To discover the longest path in the graph, we first identified the largest connected component using the components() function, assuming the longest path would reside within this densely connected area. This component formed the basis of a subgraph, where we assigned degree attributes to each node. Leveraging this structure and the power of Depth-First Search (DFS), we were able to calculate the longest distance between any two vertices. Our analysis revealed that the longest path stretches 9955 units between nodes 61710 and 67561.

```
> Subgrah_v2<-induced_subgraph(g,which(igraph::components(g)$membership==1))
```

```
> V(Subgrah_v2)$degree=igraph::degree(Subgrah_v2)
> Res=dfs(Subgrah_v2,root=1,dist=TRUE)$dist
> sort(Res,decreasing = TRUE)
61710 67561 36935 61709 61711 63401 63893 38748 69949 57193 61679
 9955  9955  9954  9954  9954  9954  9954  9953  9953  9953  9953
61680 61681 61682 41214 41212 41213 32077 75287 24951 55548 57192
 9953  9953  9953  9953  9953  9953  9952  9952  9952  9952  9952
59268 59269 59582 59583 60676 60677 68358 68359 61180 61687 61688
 9952  9952  9952  9952  9952  9952  9952  9952  9952  9952  9952
67355  4166  9044 32087 49257 24360 30164 55547 32139 39283 57788
 9952  9951  9951  9951  9951  9951  9951  9951  9951  9951  9951
57789 28171 59581 61099 28673 62459 67773 30931 13882 25170 46825
 9951  9951  9951  9951  9951  9951  9950  9950  9950  9950  9950
32078 32079 25244 15469 54098 18327 55830 55831  6962 20397 34901
 9950  9950  9950  9950  9950  9950  9950  9950  9950  9950  9950
61686 66399 66400 67158 67159  4168 75388 32085 25235 49256 14037
 9950  9950  9950  9950  9950  9949  9949  9949  9949  9949  9949
20036 55546 57804 57805 57806 57808 57809 57810 57811 57812 58535
 9949  9949  9949  9949  9949  9949  9949  9949  9949  9949  9949
59411 60955 32084 61393 66333 66334 66335 15875 75727 32074 20439
 9949  9949  9949  9949  9949  9949  9949  9948  9948  9948  9948
49237 30721 50441 54651 24757 29103 30819 55076 55077 55078 55079
 9948  9948  9948  9948  9948  9948  9948  9948  9948  9948  9948
55080 55081 56359 56360 56361 56363 56364 56365 56366 56367 56368
 9948  9948  9948  9948  9948  9948  9948  9948  9948  9948  9948
56369 56370 56371 56372 56374 56375 21621 22526 61100 61101 61102
 9948  9948  9948  9948  9948  9948  9948  9948  9948  9948  9948
61103 61464 64898 64900 43641 31985 25241 29933 46301 29996 16889
 9948  9948  9948  9948  9947  9947  9947  9947  9947  9947  9947
30827 48712 31983 14774 55490 31987 61134 65021 31113 66456 15382
 9947  9947  9947  9947  9947  9947  9947  9947  9947  9947  9946
43929 49221 45440 29995 32552 46925 46926 46927 46928 52471 53470
 9946  9946  9946  9946  9946  9946  9946  9946  9946  9946  9946
39160 30833 55471 55864 55866 64331 64332 66688 42982  4339 44536
 9946  9946  9946  9946  9946  9946  9946  9946  9945  9945  9945
15369 15949 46168 46174 46177 33572 11046  4508 55728   957 57688
 9945  9945  9945  9945  9945  9945  9945  9945  9945  9945  9945
57689 31984 60396 64349 64351 31474 67521 36942 42649 43080 44105
 9945  9945  9945  9945  9945  9945  9945  9944  9944  9944  9944
44537  3427 20103 27301 74604 52472 54195 55036 56793 57786 28296
```

d. Largest Clique(s)

Our analysis focused on identifying the largest cliques within the graph g using the largest_cliques() function. This function revealed that the largest clique requires all its members to have a minimum degree of 23, as shown in the results. While this approach directly identifies the largest cliques, an alternative method using binary search and graph splitting can also be employed to achieve the same outcome.

```
> largest_cliques(g)
[[1]]
+ 23/75879 vertices, named, from 7e48f82:
 [1] 26109 45553 75103 38109 226   44441 68882 15553 337   22220
[11] 49998 56661 2111  2556  74770 31442 2222  1111  50109 38775
[21] 75547 17775 21108

[[2]]
+ 23/75879 vertices, named, from 7e48f82:
 [1] 26109 45553 75103 38109 226   44441 68882 15553 337   22220
[11] 14776 56661 75436 1889  2556  17775 38775 75547 21108 50109
[21] 59883 74770 62661

[[3]]
+ 23/75879 vertices, named, from 7e48f82:
 [1] 26109 45553 75103 38109 226   44441 68882 15553 337   22220
[11] 14776 56661 75436 1889  2556  17775 38775 75547 21108 50109
[21] 59883 74770 2222

[[4]]
+ 23/75879 vertices, named, from 7e48f82:
 [1] 26109 45553 33443 68882 38109 226   44441 2222  337   22220
[11] 15553 56661 2111  31442 1111  17775 49998 50109 21108 74770
[21] 75547 2556  38775

[[5]]
+ 23/75879 vertices, named, from 7e48f82:
 [1] 26109 45553 33443 68882 38109 226   44441 2222  337   22220
[11] 15553 56661 1889  14776 75436 38775 21108 17775 50109 2556
[21] 75547 74770 59883

[[6]]
+ 23/75879 vertices, named, from 7e48f82:
 [1] 26109 45553 33443 68882 38109 27775 74770 2556  44441 2222
[11] 31442 50109 337   22220 15553 17775 56661 2111  21108 38775
[21] 1111  75547 49998
```

```
[[13]]
+ 23/75879 vertices, named, from 7e48f82:
 [1] 71104 44441 75103 226   38109 18886 68882 448   2222  1889
[11] 337   22220 7221  15553 44442 14776 49997 62550 33554 1778
[21] 19997 69993 75436

[[14]]
+ 23/75879 vertices, named, from 7e48f82:
 [1] 71104 44441 75103 226   38109 18886 68882 56661 337   15553
[11] 33554 2222  75436 1889  14776 22220 62550 49997 44442 69993
[21] 7221  1778  19997

[[15]]
+ 23/75879 vertices, named, from 7e48f82:
 [1] 71104 44441 33443 18886 38109 226   68882 337   2222  22220
[11] 1889  448   75436 14776 7221  62550 69993 19997 15553 1778
[21] 33554 44442 49997

[[16]]
+ 23/75879 vertices, named, from 7e48f82:
 [1] 71104 44441 33443 18886 38109 226   68882 337   2222  22220
[11] 1889  56661 15553 69993 75436 14776 33554 62550 7221  49997
[21] 44442 17775 42219

[[17]]
+ 23/75879 vertices, named, from 7e48f82:
 [1] 71104 44441 33443 18886 38109 226   68882 337   2222  22220
[11] 1889  56661 15553 69993 75436 14776 33554 62550 7221  49997
[21] 44442 1778  42219

[[18]]
+ 23/75879 vertices, named, from 7e48f82:
 [1] 71104 44441 33443 18886 38109 226   68882 337   2222  22220
[11] 1889  56661 15553 69993 75436 14776 33554 62550 7221  49997
[21] 44442 1778  19997

Warning message:
In largest_cliques(g) :
  At vendor/cigraph/src/cliques/maximal_cliques_template.h:219 : Edge directions are ignored for maximal clique calculation.
```

e. Ego(s)

The function ego(g) calculates the neighborhoods of the given vertices with the given order parameter.

```
> Ego_Of_Graph_g<-igraph::ego(g)
> Ego_Of_Graph_g
```

```
[[886]]
+ 11/75879 vertices, named, from 7e48f82:
 [1] 22872 5733  54498 55710 2     26054 52864 61428 61417 40617 58671

[[887]]
+ 362/75879 vertices, named, from 7e48f82:
  [1] 22943 925   2501  3111  3778  5289  5766  6000  7800  8705  9831  9997  10553 10886 11588 11688 12221 14276 14998 15442 16631 19443
 [23] 21220 23032 27775 28886 33331 34776 37021 39742 40864 42331 47876 52886 54531 61106 63327 64438 67772 73248 73281 73436 74214 74773
 [45] 75559 659   2012  2745  3136  4923  5001  5733  7133  7510  10982 13298 15065 16265 24154 26998 27799 27998 28221 28664 28775 29442
 [67] 33410 35998 36887 39109 40664 41143 43787 45065 46265 47121 47354 49998 51576 51986 54297 57317 62550 65438 65994 69216 70882 71427
 [89] 74042 75292 75332 2     1978  2668  50997 65105 5833  13532 19032 21465 30165 70105 73459 73481 74536 27354 59906 33153 73807 4401
[111] 4812  7911  22910 25287 44097 45899 47476 74063 668   13499 20709 25332 32208 74348 75614 24320 2085  11477 13932 16642 35221 39953
[133] 5789  6744  7447  11421 64662 73584 74112 35409 40631 73392 74789 404   21731 75403 25232 33355 57584 75337 15243 66283 12932 17031
[155] 11765 4579  18232 59095 75487 66239 41776 16398 18121 7388  16799 51998 56217 73948 26721 5445  6988  25298 21031 25265 2845  14477
[177] 25209 37132 37298 75221 37087 9986  18098 3512  18176 27021 35520 64850 66094 39565 73284 1620  10065 13621 25254 43776 67439 8309
[199] 5589  67818 6844  68199 74148 56484 18254 36888 15999 73970 5622  5877  7044  14532 47320 53698 60218 64327 73359 5822  48809 74230
+ ... omitted several vertices

[[888]]
+ 126/75879 vertices, named, from 7e48f82:
  [1] 23332 115   371   1445  1889  2000  2222  2445  3000  4556  5000  5777  5999  6554  7221  9831  9997  10220 10553 12221 12776 13331
 [23] 14187 14665 16664 17775 19886 20553 20742 21108 21476 22220 23331 25553 26664 27587 31664 33330 33331 35665 36331 36664 38109 42219
 [45] 48886 49886 50553 54441 60550 62661 64438 64772 73770 73992 74992 75103 160   326   836   1856  1989  23554 27220 31331 33332 34220
 [67] 34887 34998 36220 36442 38553 38775 39442 39998 41442 41875 43553 49553 51442 69216 69438 74326 74706 2     2123  68994 75592 10365
 [89] 54219 74536 10875 19942 10831 471   23243 4701  16398 45577 54886 56217 73682 1812  16431 9887  1823  4045  16087 17987 24232 73581
[111] 19920 74699 537   75525 33532 671   53787 3612  10221 74705 74701 28910 74702 39898 74700 74704

[[889]]
+ 121/75879 vertices, named, from 7e48f82:
  [1] 23432 780   1778  2445  3111  4000  6855  7266  8886  10553 11109 12776 16631 17442 18886 19997 20220 24899 26664 27265 29997 34776
 [23] 36331 37021 48886 52886 55553 59328 60550 73248 74770 271   939   1989  7133  7666  13843 21909 33710 33887 34443 36220 36810 38775
 [45] 38887 38998 39431 43787 51376 51986 74747 2     25887 215   3023  6377  21465 51243 51431 75581 6877  7188  10875 28687 61772 3078
 [67] 8531  24187 7244  14743 73548 3200  35121 55208 3590  11677 2756  52020 70028 49709 349   18176 49198 56206 68883 13621 67818 21787
 [89] 73359 60772 7811  14987 3187  19487 26065 65216 20653 25354 27376 21875 21953 48587 24132 3282  21988 7155  73235 45254 9873  48121
[111] 4291  21520 49176 21587 729   16071 21985 2147  21987 21989 21990
```

```
[[996]]
+ 228/75879 vertices, named, from 7e48f82:
  [1] 32220 182   780   2111  5000  5367  7800  10775 19886 24821 26431 27265 29220 37775 45998 57772 59328 60550 61105 73547 74770 75548
 [23] 8642  13298 19842 21975 24776 27331 34220 34665 37888 39687 40664 43108 46387 47331 51442 70882 71919 74092 2     41121 6377  55441
 [45] 55552 68222 2779  75581 7999  71    46620 69450 8221  66549 55086 66416 75784 75835 35121 25732 54820 20554 54931 13677 13965 8198
 [67] 33743 40964 13666 61750 75126 74148 73311 10021 75831 45443 38376 64327 74281 9953  6833  11111 13999 37221 44999 75764 593   75852
 [89] 65216 60506 10254 73299 75767 75860 33532 53131 6822  63539 75773 75810 74981 75857 54020 17465 20187 75854 16143 73693 73499 44987
[111] 75842 991   75771 75859 13011 75845 75765 59007 30021 75794 36631 75782 75818 75749 75844 73808 75849 75766 70550 60883 72005 75841
[133] 75371 75850 75768 75809 75824 27698 75856 75798 75861 75780 73757 75853 75811 75756 16209 75796 75512 75761 21347 75760 22274 75808
[155] 75828 75802 75375 30752 67501 69241 69266 71914 71915 71917 71918 71920 74758 75750 75751 75752 75753 75754 75755 75757 75759 75762
[177] 75763 75772 75774 75775 75776 75777 75778 75779 75783 75785 75786 75787 75788 75789 75790 75791 75793 75795 75797 75799 75800 75801
[199] 75804 75805 75806 75807 75812 75813 75815 75816 75817 75819 75820 75821 75822 75823 75826 75827 75829 75830 75832 75833 75834 75837
+ ... omitted several vertices

[[997]]
+ 53/75879 vertices, named, from 7e48f82:
 [1] 32331 4222  5000  8886  13554 19886 20886 23331 33330 33776 44441 63327 73658 326   24220 30442 34554 37998 45331 2     75647 55552
[23] 4457  93    70993 1734  66549 16043 1434  73392 73614 75571 1401  74118 48431 24265 66516 9887  75059 60    24466 20809 28498 72771
[45] 1512  73213 74314 73642 28409 75634 74355 40243 75862

[[998]]
+ 35/75879 vertices, named, from 7e48f82:
 [1] 32442 7800  8886  12187 12788 15331 15921 20220 58883 16265 16332 25442 70549 2     2112  35409 47365 25398 11743 35132 12444 74636
[23] 9953  7599  31409 36221 43942 62806 75176 74615 54620 75864 26987 45599 75863

[[999]]
+ 978/75879 vertices, named, from 7e48f82:
  [1] 32487 1334  2778  2889  3111  3145  3334  4778  5778  6855  7665  12388 13065 14276 19997 21109 21553 22332 31664 32576 36553 37775
 [23] 42031 43330 47876 48886 49886 54441 59994 61106 67051 67106 73281 74103 83    326   354   939   1337  1375  1589  1750  2301  4211
 [45] 4531  4801  4890  5001  5056  5145  5278  5655  5678  6889  6955  9431  10132 10965 12987 18376 18932 24154 24443 27331 27799 31920
 [67] 35554 36887 36998 39409 44197 44330 47121 49553 54498 55264 55331 57473 70039 70549 70882 71305 72550 73249 74212 74443 74747 74867
 [89] 74925 2     2056  2112  74870 3501  9054  15209 17854 65883 75581 2232  11888 14643 55020 61784 74970 75847 1952  28032 29498 75479
[111] 7954  37954 1689  33943 35221 75138 2013  73485 16043 60628 69105 74735 2608  7088  7099  7458  7460  7462  9243  55086 73326 71038
[133] 18032 29831 8343  15566 21731 27487 54875 9043  74993 4545  34232 61262 567   31742 46843 55242 227   3200  4119  7108  8076  9387
[155] 14329 14447 15773 18365 18720 30032 34699 35365 44264 45932 54909 61240 61795 62972 63795 1386  15767 18232 53653 68771 74725 38864
[177] 7508  1975  4756  6281  11127 14309 18198 19977 18121 64550 4501  3545  3472  4512  6605  8000  10055 11002 11741 12567 14393 14479
[199] 16192 30150 34320 34332 37409 38002 57417 74451 74530 5445  74433 75097 283   6180  6988  9454  11425 16343 24488 44031 44153 57395
+ ... omitted several vertices

[[1000]]
```

```
[[1000]]
+ 24/75879 vertices, named, from 7e48f82:
 [1] 32553 448   1889  2222  5000  7221  9442  13331 26664 44442 60550 2     6621  238   3845  73270 3923  18132 3335  3579  44364 19631
[23] 31042 75865

 [ reached getOption("max.print") -- omitted 74879 entries ]
> i
```

For each vertex, we computed the egos, but omitted the majority of the result values.

f. Power Centrality

Power centrality in network analysis evaluates a node's influence by considering not just its direct connections but also the influence of its connected neighbors. It reflects the idea that being connected to influential nodes elevates a node's importance within the network, capturing the concept of influence through association.

```
> power_central_graph<-power_centrality(Subgraph_Duplicate,exponent=0.5)
> sort(power_central_graph,decreasing = TRUE)
    37420     27758     25665     43887     35365     25398     74159     32798       393      7881     74969      1378      4151
1.0920500 0.8736400 0.6552300 0.6552300 0.6552300 0.6552300 0.6552300 0.6552300 0.6552300 0.6552300 0.6552300 0.6552300 0.6552300
    73093      3984     41075     43700     43702     43722     34007     26387     10258     26970     10627     74797     19961
0.6552300 0.6552300 0.6552300 0.6552300 0.6552300 0.6552300 0.6552300 0.6552300 0.6552300 0.6552300 0.6552300 0.6552300 0.6552300
    74186     62241      5987     18032     23332     74170     45987     68849     26010     42708     18943      5023     27665
0.6552300 0.6552300 0.6552300 0.4914225 0.3276150 0.3276150 0.3276150 0.3276150 0.3276150 0.3276150 0.3276150 0.3276150 0.3276150
    61199      9072     11578     14298     16331     16666     16998     17998     18831     23998     27676     55464     64839
0.3276150 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
     2368      3300     14030     15443     18999     55441     71605      7422     45121      7430     39953       530     74100
0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
      835      1058     74145       567      1401      1323      2923     54886     20554      5807     20593      1426     74017
0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
    21097     35721     57796     28243     41949      7575      4674     73320     41992      1842     12688     73401     68918
0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
    17487     43587      1078     13098     73355     19175     42057      7740     37354     52775     18164     36854     66472
0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
    17287     47832     39087     42122     42149      9964      7842     42332        63     42242     20931     42257      1193
0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
     6610     73786      4255     26674     17555       486     31418     32142      6264     74909     11211     23863      1341
0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
    12855      2600     42495     42499     42511     29320     10651       923      6429     48032     25211       717      7945
0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
    12177     15259     33409     66842     22699      7956     42734     42741     75275     42838     22497     14782     72147
0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
    49209     23055     43095     43125     43141     25715     45487     43260      9327     24567      8083     27538     43411
0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
     1285     20104     75355     10414     15676      9843      5393      7080     43624     43634     19397     75565     10962
0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
    43666     37721     19964     18841     43747     39964     66785     73372      3675     35683     43881     23553     51110
0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
    35308     44147     18668      7121     11889      8268     56962     44233     27040     23954     15896     57940     44258
0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
    44318     22029      8324      1606     53250     73487     44377     44404     18096     73652      2541     27806     31875
0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
    58184     67706     44470     44479     75239     16513     44508     10662      1009      8505     73870     44549      3420
0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
    33822      3193     44591     44669     44686     38119     44803      8650      8686     44816      8738     44840     23734
0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
    27627      8761     44923     70384     23267     65595     45081      8969     21333     45173     60362     74404     10805
```

5. Discussion

This project provided a comprehensive exploration of graph analysis techniques within the R programming environment, with a specific focus on working with medium and large-scale datasets. We gained practical experience by not only learning how to efficiently generate graphs from these datasets but also by delving into the functionalities of specialized R packages such as igraph and sna. This hands-on approach extended to crafting our own functions, further solidifying our understanding of R's capabilities and fostering independent problem-solving skills.

Beyond graph creation, the project emphasized the importance of effective communication through data visualization. We explored various techniques, including charting, visual representations, and parameter adjustments, to transform complex graphs into comprehensible formats. This enhanced our ability to interpret and analyze intricate network structures. Furthermore, the project exposed us to a diverse set of graph analytics tools, equipping us with the knowledge to delve deeper into network analysis. By studying crucial metrics like power centrality, longest paths, cliques, and alpha centrality, we were able to identify and understand hidden patterns, connections, and structural characteristics within the data. We even explored the application of random walks as a valuable tool for identifying communities within the graph, providing deeper insights into its underlying network structure.

Through the culmination of this project, we have not only gained a comprehensive understanding of graph analysis techniques and R programming but also fostered confidence in our ability to effectively handle large-scale datasets, analyze and visually represent complex network structures, and extract valuable information through the utilization of diverse tools and metrics. This newfound expertise equips us to tackle future challenges involving network analysis and data visualization with greater proficiency and insight.