

# **CSCI 6444 : INTRODUCTION TO BIG DATA & ANALYTICS**

## **Class Project 1 – Graph Analytics**

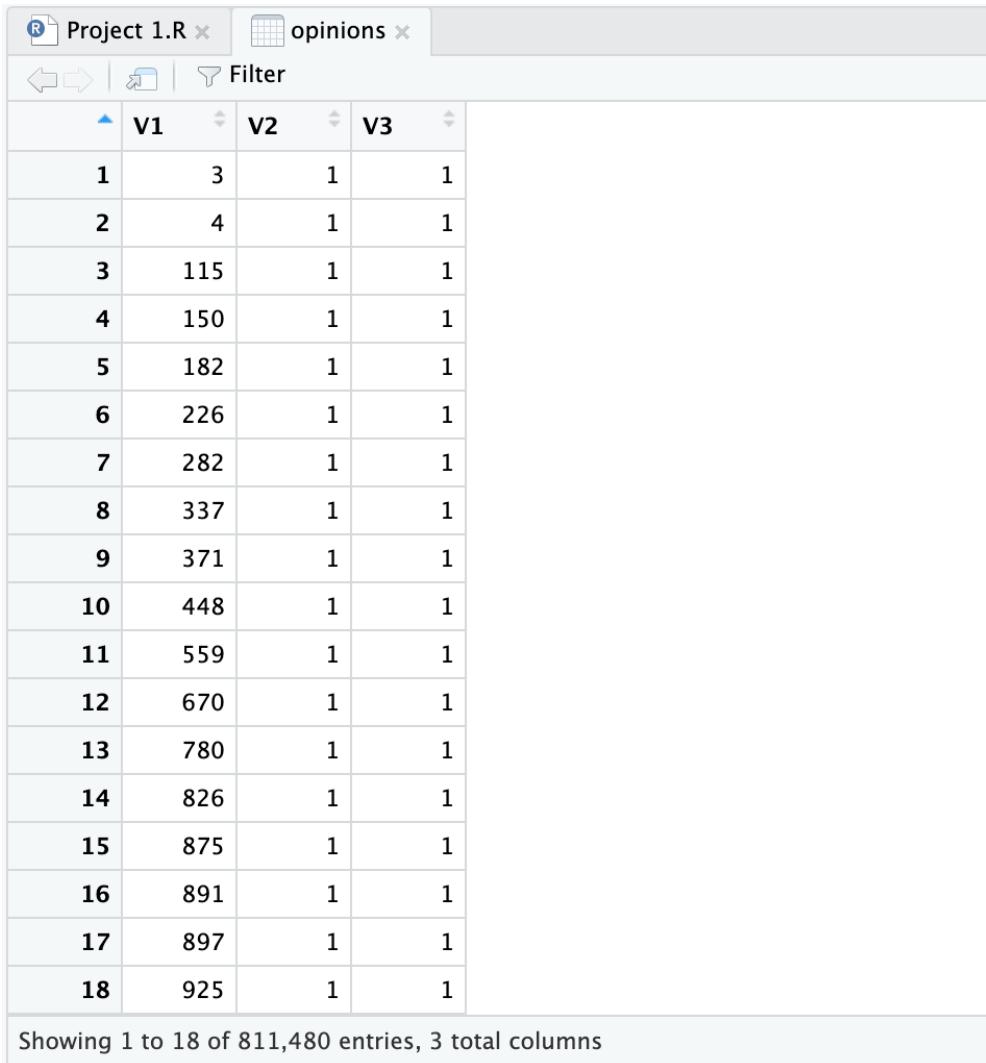
**Professor: Stephen Kaisler**

**Group 20**

**Yashwanth Raj Varadharajan [G47635180]  
Poojasri Mothukuri [G46773587]**

## Dataset

The given dataset [soc-Epinions1\_adj.tsv] is a graph of links of opinions from the SOC-E website. There are total of 811480 entries in the dataset and approximately 10 million nodes. Format of each row is <node-1>, <node-2>, # Edges.



	V1	V2	V3
1	3	1	1
2	4	1	1
3	115	1	1
4	150	1	1
5	182	1	1
6	226	1	1
7	282	1	1
8	337	1	1
9	371	1	1
10	448	1	1
11	559	1	1
12	670	1	1
13	780	1	1
14	826	1	1
15	875	1	1
16	891	1	1
17	897	1	1
18	925	1	1

Showing 1 to 18 of 811,480 entries, 3 total columns

## Installing igraph package from one of CRAN mirrors.

The igraph package is used to generate graphs, compute centrality measures and path length based properties as well as graph components and graph motifs. We use the install.package command to install igraph package from CRAN servers.

```
R 4.3.2 · ~/Documents/GWU/Sem 2/Intro to Big Data/Project/Project 1/R Project/
> install.packages("igraph")
trying URL 'https://cran.rstudio.com/bin/macosx/big-sur-x86_64/contrib/4.3/igraph_2.0.2.tgz'
Content type 'application/x-gzip' length 10317315 bytes (9.8 MB)
=====
downloaded 9.8 MB

The downloaded binary packages are in
  /var/folders/pr/hwfrw1gn4k3bbnz1vngy9380000gn/T//Rtmpm6CMk2 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
```

It is important that once we download and install the package, we import the package into our program. To do this we have used the library command.

## Importing the specified dataset

In this step we import the dataset into our program. Only then will we be able to perform the necessary operations. To achieve this goal we use certain functions.

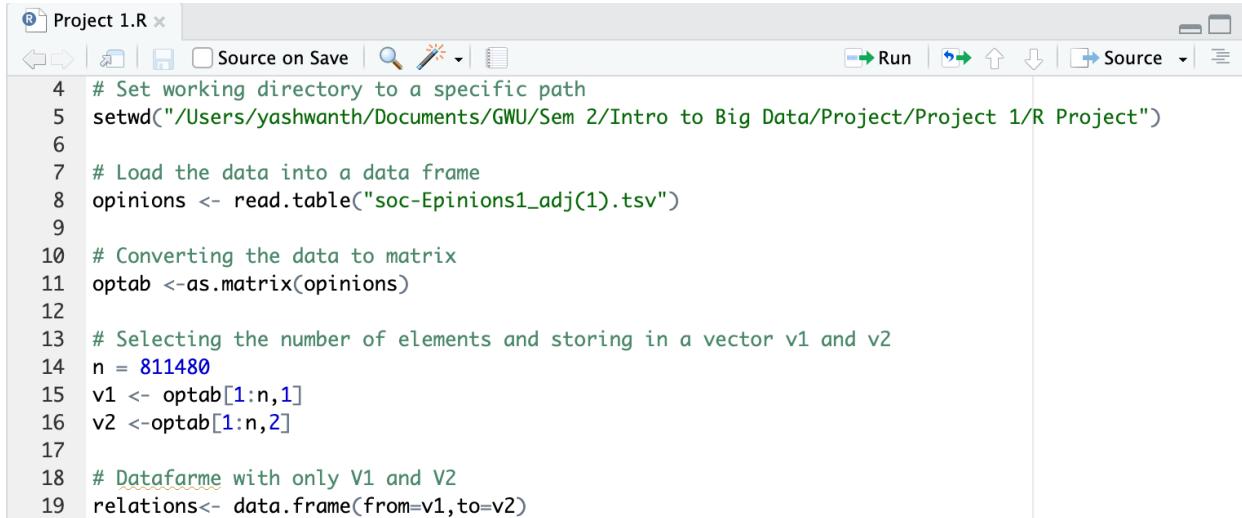
**setwd** - This function specifies the present working directory from which all files are to be taken. Inside this specified directory is where the dataset is stored.

**read.table** – This function reads the data from the file and stores it in a variable called opinions. The read.table function is used to read tabular data from text files.

**as.matrix** – This function converts the data in opinions to a matrix and stores it in a variable called optab. Matrices are a fundamental data structure in R that can be used to store and manipulate numerical data.

**data.frame** – This function is used to create a dataframe, which is a fundamental data structure for organizing and manipulating data. Dataframes are rectangular data structures where

each column can be of a different data type and each row represents a separate observation or record.



```

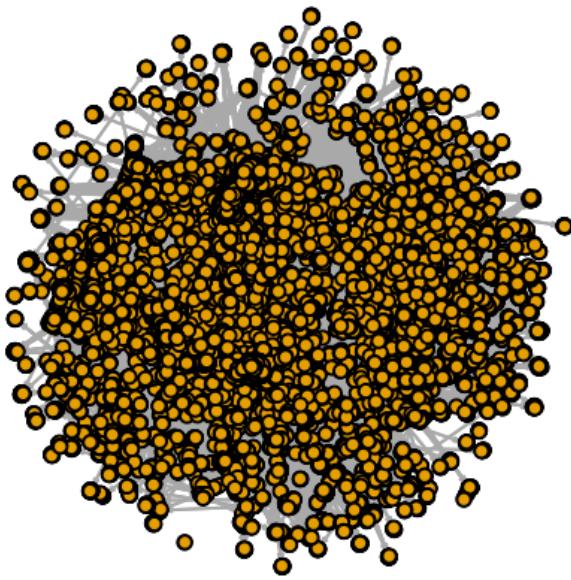
4 # Set working directory to a specific path
5 setwd("/Users/yashwanth/Documents/GWU/Sem 2/Intro to Big Data/Project/Project 1/R Project")
6
7 # Load the data into a data frame
8 opinions <- read.table("soc-Epinions1_adj(1).tsv")
9
10 # Converting the data to matrix
11 optab <- as.matrix(opinions)
12
13 # Selecting the number of elements and storing in a vector v1 and v2
14 n = 811480
15 v1 <- optab[1:n,1]
16 v2 <- optab[1:n,2]
17
18 # Dataframe with only V1 and V2
19 relations<- data.frame(from=v1,to=v2)

```

## Creating and Plotting Graph

After using these functions, the relations dataframe is created using the vectors v1 and v2. Each row in the relations dataframe represents a relationship between two entities. The relations dataframe organizes the relationship data in a structured format, which can be useful for further analysis, visualization, or modeling tasks. It provides a convenient way to work with relationship data and can be input to functions or algorithms that require relationship data in this format.

We make use of **graph\_from\_data\_frame()** function to create a data frame and then a graph. Once data frame is created, we move forward towards plotting the graphs. We make use of **plot** function and define the parameters as given in the snippets.



## Applying functions to the given dataset

Following the rubric, we apply the functions in the Introduction to Graph Analytics document and describe the results as screenshots, along with a description what the results can tell us about the problem domain.

### Vertices

The vertices of a graph are returned by the **V(g)** function, and we can observe that graph g has 75879 vertices. Here, we see labels for the nodes since in this collection, names have been eliminated and replaced with numbers.

```
> # Determining the. vertices of a graph
> V(g)
+ 75879/75879 vertices, named, from ca72a26:
 [1] 3     4     115   150   182   226   282   337   371   448   559   670   780   826   875   891
[17] 897   925   1002  1111  1112  1122  1223  1334  1445  1556  1667  1778  1834  1889  2000  2001
[33] 2080  2111  2149  2222  2223  2242  2334  2346  2434  2445  2501  2534  2556  2601  2623  2667
[49] 2727  2778  2811  2889  3000  3041  3089  3110  3111  3145  3222  3305  3333  3334  3379  3410
[65] 3445  3534  3556  3574  3667  3778  3889  3989  4000  4111  4158  4222  4333  4341  4444  4445
[81] 4556  4563  4667  4778  4889  4978  5000  5111  5222  5289  5333  5367  5385  5444  5456  5489
[97] 5554  5555  5556  5633  5666  5667  5744  5766  5777  5778  5804  5888  5911  5922  5999  6000
[113] 6110  6155  6221  6244  6266  6332  6346  6355  6443  6554  6665  6666  6711  6777  6789  6855
[129] 6888  6922  6999  7110  7221  7266  7332  7443  7444  7554  7665  7776  7777  7800  7888  7965
[145] 7998  8109  8220  8331  8442  8553  8664  8691  8705  8775  8789  8886  8887  8998  9010  9072
+ ... omitted several vertices
```

## Edges

The **E(g)** function returns the edges of a graph, and in our graph g we have 811480 edges. This gives us more information about the dataset.

```
> # Determining the edges of a graph
> E(g)
+ 811480/811480 edges from ca72a26 (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
[13] 780 ->1 826 ->1 875 ->1 891 ->1 897 ->1 925 ->1 1002->1 1111->1 1112->1 1122->1 1223->1 1334->1
[25] 1445->1 1556->1 1667->1 1778->1 1834->1 1889->1 2000->1 2001->1 2080->1 2111->1 2149->1 2222->1
[37] 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
[61] 3333->1 3334->1 3379->1 3410->1 3445->1 3534->1 3556->1 3574->1 3667->1 3778->1 3889->1 3989->1
[73] 4000->1 4111->1 4158->1 4222->1 4333->1 4341->1 4444->1 4445->1 4556->1 4563->1 4667->1 4778->1
[85] 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
[109] 5911->1 5922->1 5999->1 6000->1 6110->1 6155->1 6221->1 6244->1 6266->1 6332->1 6346->1 6355->1
+ ... omitted several edges
```

## Adjacency

The adjacency matrix for a graph is returned by the **get.adjacency()** method. This is our graph g's adjacency matrix, which measures 75879 x 75879. We can determine the density of our graph and even locate any isolated vertices with the aid of the adjacency matrix.

```
> g.adj = igraph::get.adjacency(g)
Warning message:
`get.adjacency()` was deprecated in igraph 2.0.0.
  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.
> g.adj
75879 x 75879 sparse Matrix of class "dgCMatrix"
[[ suppressing 17 column names '3', '4', '115' ... ]]
[[ suppressing 17 column names '3', '4', '115' ... ]]

3 . 1 . . 1 1 . 1 . . . . . . .
4 . 1 . . . 1 . 1 . 1 1 . 1 . .
115 . . . . . 1 . 1 1 . . . 1 . .
150 . . . . . . . . . . . . .
182 1 . . . . 1 1 1 . . . . 1 . .
226 1 1 . . . . 1 1 1 . . 1 . 1 .
282 .
337 1 1 1 . 1 1 . . 1 . 1 . 1 . .
371 . . . 1 1 . . . . . . .
448 1 1 1 . 1 1 . . . 1 . .
559 . . 1 . . . . . . .
670 . 1 . . . .
780 . 1 . . . 1 . 1 . 1 . . .
826 . . . . . . . .
875 . . . .
891 . 1 1 . 1 1 . 1 . . . .
897 . . . . . . .
925 . . . . . 1 . 1 . .
1002 . 1 . . . 1 . 1 . . . 1 . .
1111 1 1 1 . . 1 . 1 1 . 1 1 . . 1 . .
1112 1 . . . . 1 1 . 1 . . . .
1122 . . . . .
1223 . . . 1 . . .
1334 . . . 1 . . . .
1445 . 1 . 1 . . . . 1 . 1 . 1 . .
1556 . . . . .
1667 . . . . .
1778 . 1 . . . 1 . 1 . 1 . 1 . . 1 . .
1834 . . . . 1 . 1 . . . .
```

```
..... suppressing 75862 columns and 75821 rows in show(); maybe adjust 'options(max.print= *, width = *)'
[[ suppressing 17 column names '3', '4', '115' ... ]]

41768 . . . . .
41769 . . . . .
41770 . . . . .
41772 . . . . .
7373 . . . . .
7374 . . . . .
7375 . . . . .
7376 . . . . .
7378 . . . . .
7387 . . . . .
7389 . . . . .
7397 . . . . .
7400 . . . . .
7403 . . . . .
7405 . . . . .
7409 . . . . .
7411 . . . . .
41774 . . . . .
41777 . . . . .
41780 . . . . .
41781 . . . . .
41782 . . . . .
41783 . . . . .
41784 . . . . .
41785 . . . . .
41786 . . . . .
41789 . . . . .
41790 . . . . .
41792 . . . . .
```

## Graph Density

The **graph.density()** function determines a graph's density while accounting for the graph's type. As we can see, our graph's density is 0.000140942, indicating that it is quite sparse—that is, there are very few edges in it when compared to all the possible edges.

```
> #v Density of graph
> g.density <- graph.density(g)
> g.density
[1] 0.000140942
```

## Edge Density

The **edge\_density()** function computes the density of a graph, which is defined as the ratio of the number of edges in the graph to the maximum number of edges that might exist in the graph. It is evident that, even when loops are allowed in the graph, the density remains nearly constant. This suggests that the graph is most likely devoid of loops and may even be a simple graph.

```
> g.density
[1] 0.000140942
> igraph::edge_density(g)
[1] 0.000140942
> igraph::edge_density(g, loops=T)
[1] 0.0001409401
```

## Degree

The **degree()** function is defined in several packages however here, we use the one defined in the igraph package. It returns the number of adjacent edges for each vertex.

igraph::degree(g)							
3	4	115	150	182	226	282	337
572	690	686	48	636	880	266	924
371	448	559	670	780	826	875	891
342	826	618	226	828	38	14	324
897	925	1002	1111	1112	1122	1223	1334
40	372	332	448	498	22	362	830
1445	1556	1667	1778	1834	1889	2000	2001
508	82	366	432	216	524	2014	210
2080	2111	2149	2222	2223	2242	2334	2346
30	832	26	834	542	30	24	176
2434	2445	2501	2534	2556	2601	2623	2667
252	422	1046	224	602	352	170	510
2727	2778	2811	2889	3000	3041	3089	3110
38	1366	22	396	730	36	314	36
3111	3145	3222	3305	3333	3334	3379	3410
1736	814	104	20	732	820	666	12
3445	3534	3556	3574	3667	3778	3889	3989
144	206	362	28	390	262	1296	96
4000	4111	4158	4222	4333	4341	4444	4445
2410	236	8	664	364	20	412	640
4556	4563	4667	4778	4889	4978	5000	5111

```

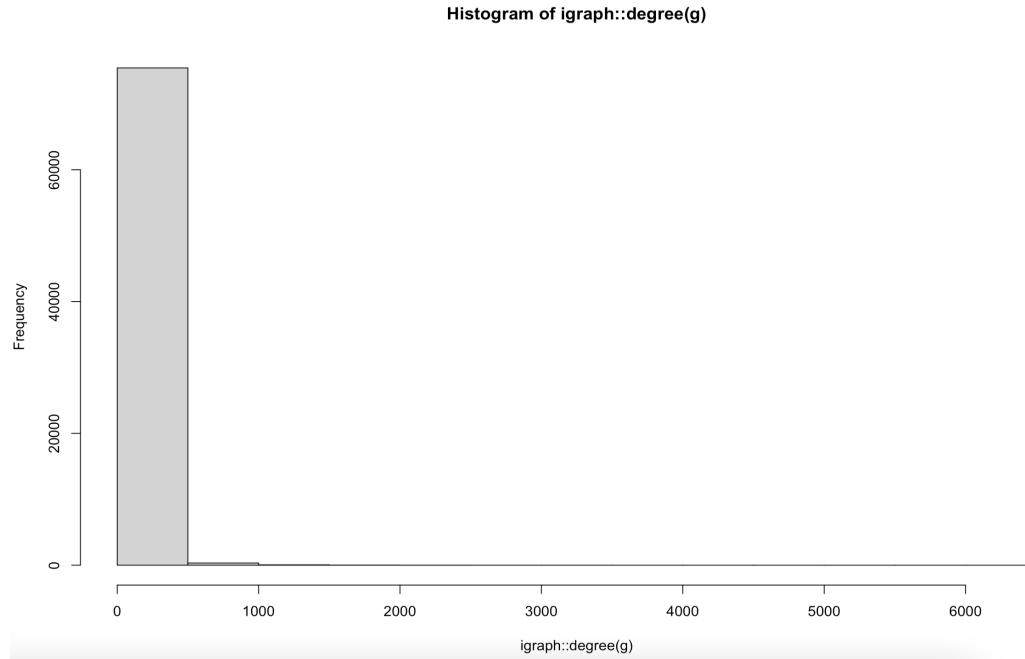
      25776 25998 26098 26109 26132 26209 26220 26262
      206   158   146   388   366   52   148   20
      26276 26331 26442 26553 26576 26665 26776 26998
      142    24    54   154   132   448   122   474
      27109 27110 27220 27331 27442 27533 27553 27664
      242   160   350   172    2    6   120   306
      27776 27799 27887 27910 27998 28109 28221 28331
      464   546    22   100   370   206   780   102
      28431 28442 28553 28653 28664 28775 28853 28887
      146   228   452   182   218   190    80   352
      28998 29109 29331 29442 29553 29664 29775 29886
      24    130   120   328   264   140   510   516
      29998 30109 30220 30331 30442 30553 30664 30775
      398    62   184   168   708   470   266   198
      30886 30997 31109 31220 31331 31352 31387 31442
      336   148   102   644   732    8   124   550
      31553 31665 31775 31831 31853 31886 31898 31920
      300   1034   348   432   316   142   168   252
      31953 31997 32108 32220 32331 32442 32487 32553
      150   888   190   454   104   68   1954   46
[ reached getOption("max.print") -- omitted 74879 entries ]

```

## Histogram

To generate a histogram and see the degrees of the graph's nodes, we utilize the **hist()** function. We may deduce from the histogram that the majority of nodes have degrees between 0 and 500, while the remaining nodes have degrees between 500 and 1000.

```
> # Histogram  
> hist(igraph::degree(g))
```



## Betweenness Centrality

The betweenness centrality of a network can be ascertained using the `centr_betw()` function. We can determine the number of shortest paths that travel through each vertex by looking at the betweenness centrality. Overall, we can observe that the nodes in our graph `g` have comparatively lower betweenness, indicating that there are fewer nodes with high betweenness (i.e., nodes that significantly impact the network's connectedness).

```
> g.between = igraph::centr_betw(g)
> g.between
$res
[1] 3.151719e+06 3.981881e+06 3.266552e+06 1.206334e+05 7.363342e+06 9.469315e+06 1.701157e+06
[8] 8.122346e+06 2.725038e+06 5.273894e+06 3.744818e+06 4.063249e+05 8.692060e+06 2.335483e+06
[15] 8.783339e+02 1.474652e+06 3.244700e+05 3.461340e+06 2.436035e+06 1.039556e+06 7.101634e+06
[22] 9.004451e+05 2.007443e+06 1.553106e+07 2.858003e+06 1.403503e+05 4.735297e+06 1.712402e+06
[29] 2.053627e+06 2.216830e+06 5.637427e+07 2.877706e+06 1.921803e+05 1.090843e+07 2.662356e+03
[36] 5.445164e+06 4.715068e+06 4.811867e+05 1.129239e+04 5.629457e+05 9.515164e+05 2.272398e+06
[43] 2.380167e+07 1.479723e+06 2.850635e+06 4.453057e+06 9.638072e+05 3.046881e+06 1.930011e+05
[50] 2.534605e+07 3.859493e+04 2.311553e+06 6.605852e+06 7.532751e+05 6.104356e+06 5.109320e+05
[57] 4.907386e+07 1.259179e+07 3.950718e+05 2.408225e+03 1.068249e+07 1.672638e+07 7.387049e+06
[64] 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
[78] 1.589372e+05 4.050165e+06 5.511125e+06 1.211066e+07 1.410818e+04 3.086957e+06 1.609299e+08
[85] 8.555053e+05 2.563614e+06 2.308962e+07 4.421859e+05 1.343681e+06 8.883725e+06 7.132626e+03
[92] 9.118853e+06 9.727240e+05 3.628466e+03 2.363156e+06 8.622036e+06 6.774917e+05 1.973869e+05
[99] 9.096726e+05 4.077454e+06 4.890626e+02 5.032214e+06 1.153062e+07 2.657635e+06 3.215569e+01
[106] 3.119489e+06 2.223556e+05 2.428348e+04 3.576354e+06 2.992492e+07 6.872374e+06 5.773731e+06
[113] 6.494667e+05 1.308536e+06 4.799707e+05 2.080087e+06 1.582126e+06 1.539632e+05 1.296378e+03
[120] 7.188281e+06 1.513014e+05 1.261128e+05 1.018983e+06 1.520621e+05 1.951136e+05 2.601437e+06
[127] 3.520919e+05 2.423314e+07 2.710753e+02 2.146971e+07 1.378418e+06 1.856139e+05 2.377946e+06
[134] 2.163510e+06 6.646553e+04 1.181188e+06 1.112392e+06 6.558604e+04 9.329885e+06 4.793107e+06
```

```
[911] 3.699075e+05 4.319918e+05 4.918855e+05 5.855767e+05 9.524471e+05 3.419097e+04 6.812148e+06
[918] 1.102528e+06 1.713223e+06 3.308208e+01 1.360030e+06 6.556471e+05 1.753463e+06 8.204644e+05
[925] 3.164228e+06 4.854558e+05 3.767764e+05 4.591628e+05 5.510034e+05 3.323746e+03 1.365638e+04
[932] 1.585927e+05 1.733604e+06 1.978938e+06 2.914033e+05 7.865149e+06 3.969551e+05 9.768958e+05
[939] 1.208389e+06 1.634546e+06 0.000000e+00 1.015647e+02 1.600146e+06 5.689131e+05 3.056968e+06
[946] 1.128253e+07 9.022492e+02 1.764749e+06 5.131798e+06 5.613639e+05 1.042252e+07 3.413214e+05
[953] 1.180260e+06 2.864004e+06 1.831078e+06 1.770580e+06 1.046662e+06 1.033573e+06 1.179021e+05
[960] 9.361194e+05 1.518796e+05 3.266932e+05 2.991570e+05 2.559759e+06 1.350596e+06 2.981400e+05
[967] 6.275796e+06 2.968178e+06 2.268620e+06 1.542616e+04 6.550544e+05 1.684846e+06 1.027345e+07
[974] 1.358577e+06 3.269994e+06 4.239067e+05 2.237957e+06 5.055153e+05 2.921046e+05 3.301058e+06
[981] 4.078889e+06 2.041223e+03 8.165402e+05 2.235724e+06 6.009564e+05 3.427375e+07 1.131710e+06
[988] 8.442873e+06 1.852482e+06 2.690190e+05 4.862595e+05 1.667332e+06 4.150323e+05 8.909589e+06
[995] 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
```

## Closeness Centrality

The closeness centrality of each graph node is returned by the **centr\_clo()** function. The node in the graph is closer to other nodes the higher its proximity value. The network g's nodes are reasonably well connected and reachable from the majority of other nodes, as evidenced by the average proximity centrality of 0.3 that we have seen. The evidence for this is also readily apparent to us thanks to our previous plots.

```
> g.closeness = igraph::centr_clo(g)
> g.closeness
$res
[1] 0.3235830 0.3182144 0.3204927 0.2823251
[5] 0.3207840 0.3293043 0.3087729 0.3229494
[9] 0.3021660 0.3246296 0.3207826 0.2975331
[13] 0.3315679 0.2943417 0.2642594 0.3140524
[17] 0.2716610 0.3160723 0.3123369 0.3177786
[21] 0.3239823 0.2667126 0.3209699 0.3424100
[25] 0.3161158 0.2923683 0.3077684 0.3178638
[29] 0.3058456 0.3158092 0.3518479 0.3082924
[33] 0.2727371 0.3334139 0.2699349 0.3197984
[37] 0.3230250 0.2749230 0.2786445 0.3031511
[41] 0.3050439 0.3111585 0.3431999 0.3161987
[45] 0.3207338 0.3198981 0.3099168 0.3268967
[49] 0.2949711 0.3353028 0.2754350 0.3199953
[53] 0.3284334 0.2871730 0.3058234 0.2739839
[57] 0.3554810 0.3386626 0.2808288 0.2706291
[61] 0.3264200 0.3364954 0.3282544 0.2685268
[65] 0.3009913 0.3027798 0.3189461 0.2903622
[69] 0.3216570 0.3056891 0.3388955 0.3063370
[73] 0.3585687 0.3120068 0.2611470 0.3222253
[77] 0.3155636 0.2713161 0.3162277 0.3303021
[81] 0.3366984 0.2834219 0.3070088 0.3623253
```

```
[945] 0.3145184 0.3305496 0.2706793 0.2972557
[949] 0.3263975 0.3060763 0.3196933 0.2980591
[953] 0.3049850 0.3216884 0.3169582 0.3056644
[957] 0.3111942 0.3041390 0.2980017 0.3049286
[961] 0.2745490 0.3032153 0.2996671 0.3035368
[965] 0.3085594 0.2990541 0.3130547 0.3215466
[969] 0.3201154 0.2888061 0.3084716 0.3004895
[973] 0.3184642 0.3216120 0.3025866 0.2936480
[977] 0.3206810 0.3072027 0.3037908 0.3250593
[981] 0.3229920 0.2702685 0.3041598 0.3224745
[985] 0.3144077 0.3419578 0.3067221 0.3210229
[989] 0.3088433 0.2847995 0.2897225 0.3112453
[993] 0.2906280 0.3286895 0.2938527 0.3077734
[997] 0.2953293 0.2978929 0.3435402 0.2820784
[ reached getOption("max.print") -- omitted 74879 entries ]
```

```
$centralization
[1] 0.7633521
```

```
$theoretical_max
[1] 75877
```

## Shortest Path Between Two Nodes

We can get the shortest path between any two nodes in a graph g using the function **shortest.paths()**. In this instance, they are shown as a matrix for the shortest route between nodes I and J. This allows us to identify more isolated nodes or vertices that are more central than others.

```
> # Shortest Path between two nodes
> g.sp = igraph::shortest.paths(newG70)
Warning message:
`shortest.paths()` was deprecated in igraph 2.0.0.
i Please use `distances()` instead.
This warning is displayed once every 8 hours.
Call `lifecycle::last_lifecycle_warnings()` to see
where this warning was generated.
> g.sp
 3 4 115 182 226 282 337 371 448 559 670 780 891
 925 1002 1111 1112 1223 1334 1445 1556 1667 1778
 1834 1889 2000 2001 2111 2222 2223 2346 2434
 2445 2501 2534 2556 2601 2623 2667 2778 2889
 3000 3089 3111 3145 3222 3333 3334 3379 3445
 3534 3556 3667 3778 3889 3989 4000 4111 4222
 4333 4444 4445 4556 4667 4778 4889 4978 5000
 5111 5222 5289 5367 5489 5554 5556 5633 5667
 5744 5766 5778 5911 5922 5999 6000 6110 6155
 6221 6244 6266 6355 6443 6554 6665 6711 6777
 6789 6855 6922 6999 7110 7221 7266 7332 7443
 7444 7554 7665 7776 7800 7965 7998 8331 8664
 8789 8886 8887 9010 9220 9442 9553 9664 9775
```

```

73288 74558 75709 23510 23721 33142 11753 21964
14097 73515 64606 10317 13488 10787 63772 1670
817 11551 39387 1108 62395 59750 44186 2487
67451 37398 67617 1232 73390 11737 4261 71082
3336 5285 61129 28709 1370 4518 23309 67506
14825 23876 8631 13719 291 10069 23743 73827
3852 13320 5125 3083 3085 16479 20889 39372
39373 41169 45504 45505 45508 73728 6366 74795
74453 6990 3094 10899 34545 45664 45667 45672
35991 17772 1643 11583 5082 7381 38487 20691
9278 73825 75621 74429 74454 73758 64217 73406
63194 63294 68627 75475 25487 9763 68224 66773
68772 74733 6648 44253 23165
[ reached getOption("max.print") -- omitted 4322 rows ]

```

## Max Clique

The igraph package's **max\_cliques()** function can be used to calculate the size of a graph's largest clique. For example, the two nodes that make up the greatest clique of node 30 are 43506 and 1317. Though not necessary to the entire graph, the existence of two cliques indicates that these nodes are intimately related to one another. Furthermore, a higher clique would represent a higher density of nodes that are connected.

```

> # Max Clique
> node <- c(30)
> g.adj_graph <- igraph::graph_from_adjacency_matrix(g.adj)
> g.30clique = igraph::max_cliques(g.adj_graph,min = NULL, max = NULL, subset = node)
> g.30clique
[[1]]
+ 2/75879 vertices, named, from 7ae63df:
[1] 43506 1317

```

## Clique num

The function **clique\_num()** determines the size of the largest clique in the graph, which is 23 in our case.

```

> # Largest clique
> g.lgclique = igraph::clique_num(g.adj_graph)
Warning message:
In igraph::clique_num(g.adj_graph) :
  At vendor/cigraph/src/cliques/maximal_cliques_template.h:220 : Edge directions are ignored for maximal clique calculation.
> g.lgclique
[1] 23

```

## Simplify() and is.simple()

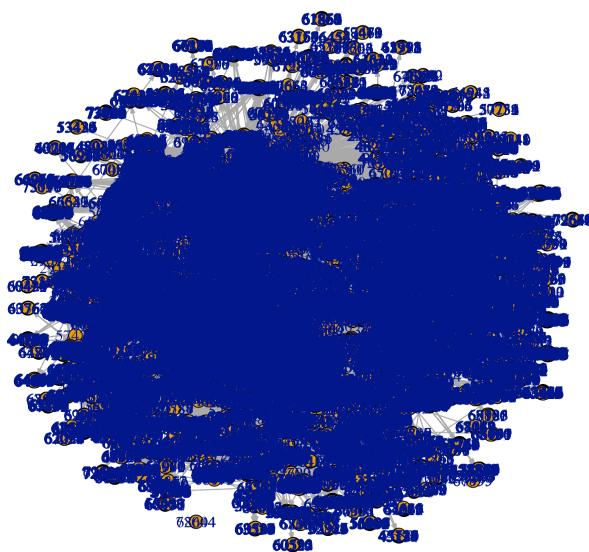
The **is.simple()** method tells us if our graph is simple i.e., if our graph has no loops and multiple edges between vertices. Without performing any simplification, we notice that our graph is

simple. This is verified when we apply the `simplify()` function on the graph after which, the `is.simple()` method still returns true.

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

## Simplification of the Graph

In the following steps we use brute force approach to simplify the graph.



Finding the vertex names

```
> # Finding the vertex names
> MyG <- igraph::V(g)
> MyG
+ 75879/75879 vertices, named, from ca72a26:
 [1] 3    4    115   150   182   226   282   337   371   448   559   670   780   826   875
 [16] 891   897   925   1002   1111   1112   1122   1223   1334   1445   1556   1667   1778   1834   1889
 [31] 2000   2001   2080   2111   2149   2222   2223   2242   2334   2346   2434   2445   2501   2534   2556
 [46] 2601   2623   2667   2727   2778   2811   2889   3000   3041   3089   3110   3111   3145   3222   3305
 [61] 3333   3334   3379   3410   3445   3534   3556   3574   3667   3778   3889   3989   4000   4111   4158
 [76] 4222   4333   4341   4444   4445   4556   4563   4667   4778   4889   4978   5000   5111   5222   5289
 [91] 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
 [121] 6443   6554   6665   6666   6711   6777   6789   6855   6888   6922   6999   7110   7221   7266   7332
 [136] 7443   7444   7554   7665   7776   7777   7800   7888   7965   7998   8109   8220   8331   8442   8553
+ ... omitted several vertices
>
```

We start the simplification process by determining the nodes with degree below 1 and 2.

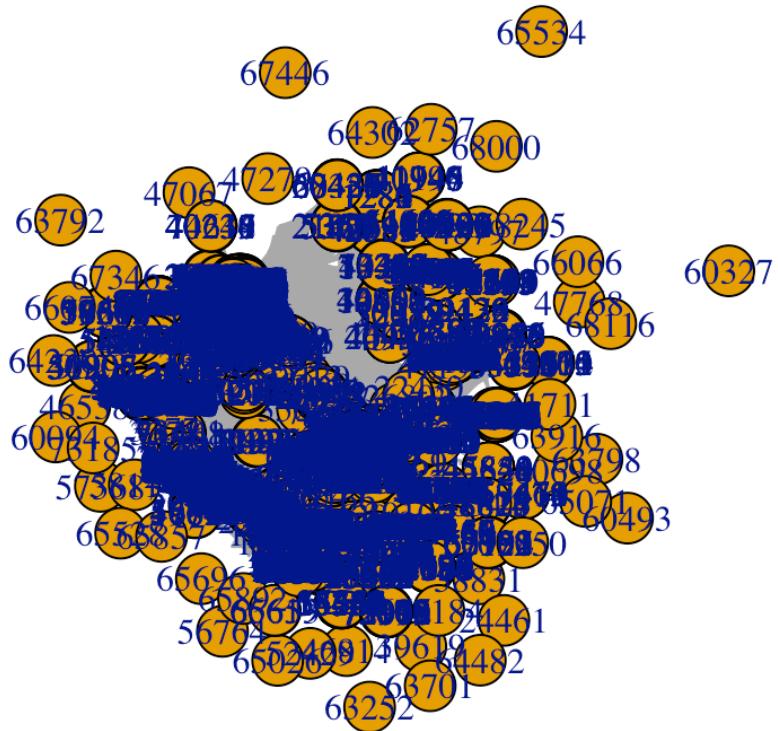
```
> # Nodes with degree 1
> MyG1 <- igraph::V(g)[igraph::degree(g)<1]
> MyG1
+ 0/75879 vertices, named, from ca72a26:
> # Nodes with degree 2
> MyG2 <- igraph::V(g)[igraph::degree(g)<2]
> MyG2
+ 0/75879 vertices, named, from ca72a26:
```

Since there are no nodes here, we go forward with determining the nodes with degree less than 10 and see if any.

```
> # Nodes with degree 10
> MyG10 <- igraph::V(g)[igraph::degree(g)<10]
> MyG10
+ 56412/75879 vertices, named, from ca72a26:
 [1] 4158   5777   7777   7888   8775   9109   10236  10664  11554  13109  13332
 [12] 14109  16998  17466  17553  18109  18867  18998  19442  19460  19754  19775
 [23] 23639  24864  27031  31575  32510  40439  49178  51108  53875  70811  70981
 [34] 72188  72189  73781  6128   8145   10961  10965  13642  16200  18246  25696
 [45] 27442  27533  31352  40220  40442  41331  41553  41775  41886  42108  42442
 [56] 42555  43219  43442  43997  44554  44776  44887  45109  45442  46665  46776
 [67] 46887  47442  48664  49109  49853  53657  65485  68311  72159  72161  72162
 [78] 72163  75028  75263  2145   2156   2257   2312   2368   2379   2446   2512
 [89] 70210  70211  70212  4435   49531  51820  51831  51875  69192  70051  70052
 [100] 7415   7417   17170  41796  41797  41799  41801  75002  41802  41803  13945
+ ... omitted several vertices
```

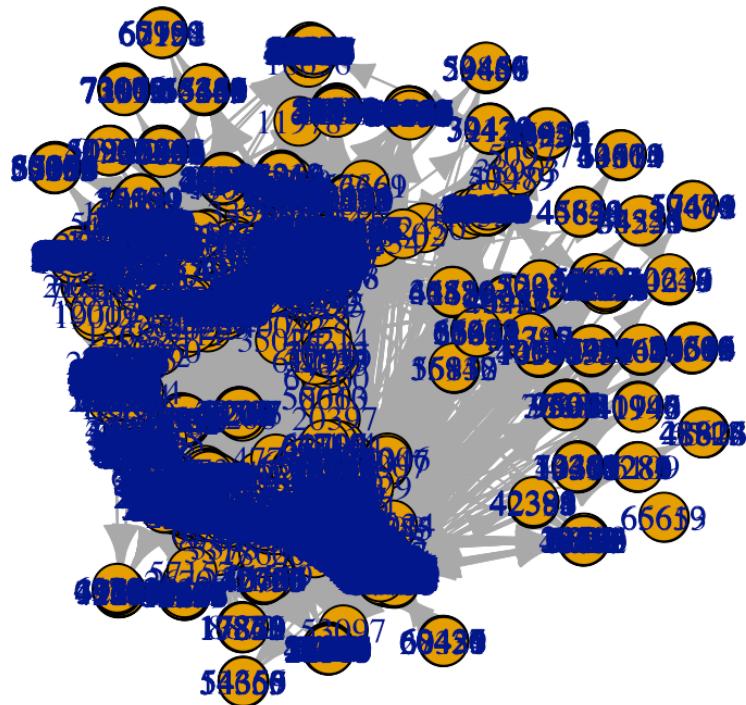
Now we go forward in removing the vertices with less than 10 edges. We store the resulting dataframe in a variable called newG.

```
> # Removing vertices with less than 10 edges
> newG <- igraph::delete.vertices(g, MyG10)
Warning message:
`delete.vertices()` was deprecated in igraph 2.0.0.
  i Please use `delete_vertices()` instead.
This warning is displayed once every 8 hours.
Call `lifecycle::last_lifecycle_warnings()` to see where this warning
was generated.
> newG
IGRAPH c05b22f DN-- 19467 649668 --
+ attr: name (v/c)
+ edges from c05b22f (vertex names):
 [1] 3    ->1 4    ->1 115   ->1 150   ->1 182   ->1 226   ->1 282   ->1 337   ->1
 [9] 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
[25] 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
[41] 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
[57] 3111 ->1 3145 ->1 3222 ->1 3305 ->1 3333 ->1 3334 ->1 3379 ->1 3410 ->1
+ ... omitted several edges
```



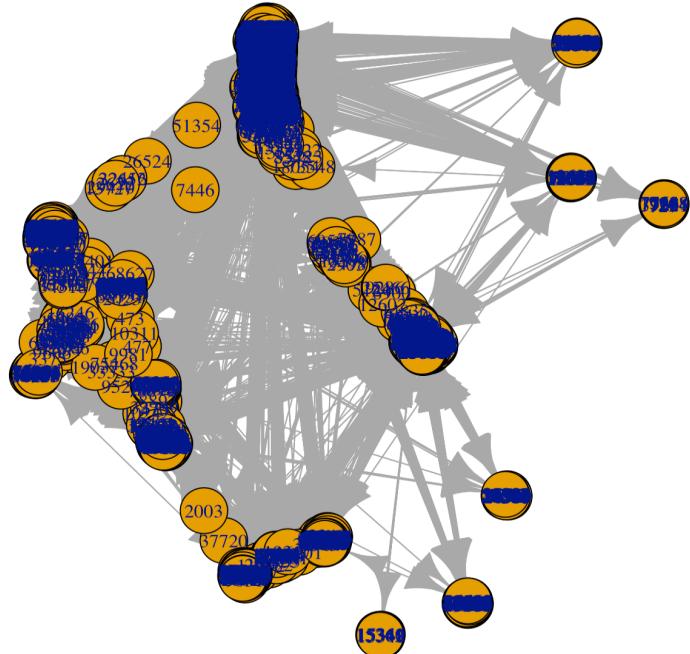
Lets now remove the vertices with degree 0 and replot.

```
> plot(newG) # Plotting
> # selecting vertices with 0 degree
> newG0 <- igraph::V(newG)[igraph::degree(newG)<1]
> newG0
+ 44/19467 vertices, named, from c05b22f:
[1] 46538 46814 47067 47279 47768 48797 39619 52429 56764 57368 58114 58245 58831 60094 60327 60493 60698
[18] 24461 62757 63252 63701 63792 63798 63916 64230 64302 64482 65026 65071 65528 65534 65696 65857 65892
[35] 66066 66074 67250 67346 67446 41711 68000 68116 73184 73185
> plot(newG0)
> # Removing vertices with degree 0
> newG2 <- igraph::delete.vertices(newG,newG0)
> newG2
IGRAPH 57e75ac DN-- 19423 649668 --
+ attr: name (v/c)
+ edges from 57e75ac (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
[13] 780 ->1 826 ->1 875 ->1 891 ->1 897 ->1 925 ->1 1002->1 1111->1 1112->1 1122->1 1223->1 1334->1
[25] 1445->1 1556->1 1667->1 1778->1 1834->1 1889->1 2000->1 2001->1 2080->1 2111->1 2149->1 2222->1
[37] 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
[61] 3333->1 3334->1 3379->1 3410->1 3445->1 3534->1 3556->1 3574->1 3667->1 3778->1 3889->1 3989->1
[73] 4000->1 4111->1 4222->1 4333->1 4341->1 4444->1 4445->1 4556->1 4563->1 4667->1 4778->1 4889->1
[85] 4978->1 5000->1 5111->1 5222->1 5289->1 5333->1 5367->1 5385->1 5444->1 5456->1 5489->1 5554->1
+ ... omitted several edges
> plot(newG2)
```



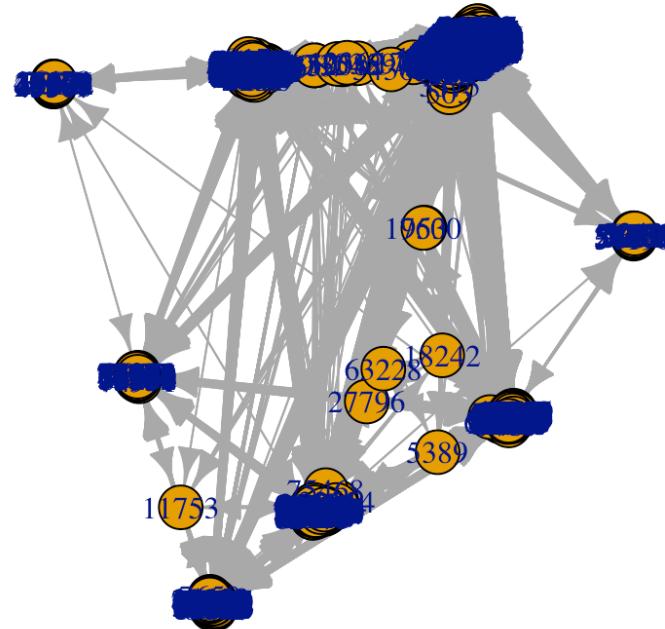
Still there is not much differentiation. Hence we go forward and remove the vertices with degree less than 30 and we increase until we get a proper simplification.

```
> # selecting vertices with degree less than 30
> newG30 <- igraph::V(newG)[igraph::degree(newG)<30]
> newG30
+ 11650/19467 vertices, named, from c05b22f:
[1] 875 1122 2080 2149 2242 2334 2811 3305 3410 3574 4341 4563 5666 5804 6332 6346
[17] 6666 6888 8109 8220 8442 8553 8691 8705 8998 9331 9646 9885 11479 11578 12109 12116
[33] 12552 13448 13776 14443 14554 15554 17331 17532 17560 17664 17950 18220 18331 18442 18664 19109
[49] 19166 19220 19331 20227 21179 21411 21997 22250 22665 22776 22781 25987 27387 29849 33231 36354
[65] 52909 54264 64839 67007 68473 72660 73647 74397 74746 75324 75411 386 513 821 919 1043
[81] 3054 4249 4391 4590 5237 6321 6814 9075 9978 10186 10799 10940 10955 10970 10973 10991
[97] 11933 12587 12742 13829 13903 16560 17352 17376 18108 18458 19916 22872 24665 24810 25311 26262
[113] 26331 27887 28998 32907 33607 39433 39664 39775 40331 40553 40886 41109 41220 41997 42886 43075
[129] 43331 43775 43886 44219 44665 45220 45665 45887 46109 46331 47109 47220 47553 47887 52379 55721
[145] 71919 73730 74172 74376 74641 74947 75698 2089 2167 2189 2200 2211 2235 2468 5150 9893
+ ... omitted several vertices
> # Removing vertices with degree less than 30
> newG30 <- igraph::delete.vertices(newG,newG30)
> newG30
IGRAPH 53c05a1 DN-- 7817 508082 --
+ attr: name (v/c)
+ edges from 53c05a1 (vertex names):
[1] 3 -> 4 -> 115 -> 150 -> 1 182 -> 1 226 -> 1 282 -> 1 337 -> 1 371 -> 1 448 -> 1 559 -> 1 670 -> 1
[13] 780 -> 1 826 -> 1 891 -> 1 897 -> 1 925 -> 1 1002 -> 1 1111 -> 1 1112 -> 1 1223 -> 1 1334 -> 1 1445 -> 1 1556 -> 1
[25] 1667 -> 1 1778 -> 1 1834 -> 1 1889 -> 1 2000 -> 1 2001 -> 1 2111 -> 1 2222 -> 1 2223 -> 1 2346 -> 1 2434 -> 1 2445 -> 1
[37] 2501 -> 1 2534 -> 1 2556 -> 1 2601 -> 1 2623 -> 1 2667 -> 1 2727 -> 1 2778 -> 1 2889 -> 1 3000 -> 1 3041 -> 1 3089 -> 1
[49] 3110 -> 1 3111 -> 1 3145 -> 1 3222 -> 1 3333 -> 1 3334 -> 1 3379 -> 1 3445 -> 1 3534 -> 1 3556 -> 1 3667 -> 1 3778 -> 1
[61] 3889 -> 1 3989 -> 1 4000 -> 1 4111 -> 1 4222 -> 1 4333 -> 1 4444 -> 1 4445 -> 1 4556 -> 1 4667 -> 1 4778 -> 1 4889 -> 1
[73] 4978 -> 1 5000 -> 1 5111 -> 1 5222 -> 1 5289 -> 1 5333 -> 1 5367 -> 1 5385 -> 1 5444 -> 1 5456 -> 1 5489 -> 1 5554 -> 1
[85] 5555 -> 1 5556 -> 1 5633 -> 1 5667 -> 1 5744 -> 1 5766 -> 1 5778 -> 1 5888 -> 1 5911 -> 1 5922 -> 1 5999 -> 1 6000 -> 1
+ ... omitted several edges
> plot(newG30)
```



We now increase degree to 70 and plot the graph.

```
> # selecting vertices with degree less than 70
> newG70 <- igraph::V(newG)[igraph::degree(newG)<70]
> newG70
+ 15145/19467 vertices, named, from c05b22f:
[1] 150   826   875   897   1122   2080   2149   2242   2334   2727
[17] 4341   4563   5333   5385   5444   5456   5555   5666   5804   5888
[33] 8442   8553   8691   8705   8998   9072   9331   9646   9885   9886
[49] 11665  11866  11887  12109  12116  12552  12625  12665  13220  13443
[65] 15554  16331  17010  17220  17331  17532  17560  17664  17950  18220
[81] 19220  19331  19553  20227  20442  21179  21411  21997  22250  22665
[97] 27676  29849  33231  33421  36354  39443  41354  42720  43053  43264
[113] 52842  52909  53398  54142  54264  54286  54642  56807  59972  63573
[129] 68118  68201  68417  68473  72660  73347  73573  73647  74397  74399
[145] 75453  386   513   545   816   821   895   919   939   1043  1646  1672
+ ... omitted several vertices
> # Removing vertices with degree less than 80
> newG70 <- igraph::delete.vertices(newG,newG70)
> newG70
IGRAPH 153b195 DN-- 4322 384422 --
+ attr: name (v/c)
+ edges from 153b195 (vertex names):
[1] 3   ->1 4   ->1 115  ->1 182  ->1 226  ->1 282  ->1 337  ->1 371  ->1 448  ->1 559  ->1 670  ->1 780  ->1
[13] 891  ->1 925  ->1 1002 ->1 1111 ->1 1112 ->1 1223 ->1 1334 ->1 1445 ->1 1556 ->1 1667 ->1 1778 ->1 1834 ->1
[25] 1889 ->1 2000 ->1 2001 ->1 2111 ->1 2222 ->1 2223 ->1 2346 ->1 2434 ->1 2445 ->1 2501 ->1 2534 ->1 2556 ->1
[37] 2601 ->1 2623 ->1 2667 ->1 2778 ->1 2889 ->1 3000 ->1 3089 ->1 3111 ->1 3145 ->1 3222 ->1 3333 ->1 3334 ->1
[49] 3379 ->1 3445 ->1 3534 ->1 3556 ->1 3667 ->1 3778 ->1 3889 ->1 3989 ->1 4000 ->1 4111 ->1 4222 ->1 4333 ->1
[61] 4444 ->1 4445 ->1 4556 ->1 4667 ->1 4778 ->1 4889 ->1 4978 ->1 5000 ->1 5111 ->1 5222 ->1 5289 ->1 5367 ->1
[73] 5489 ->1 5554 ->1 5556 ->1 5633 ->1 5667 ->1 5744 ->1 5766 ->1 5778 ->1 5911 ->1 5922 ->1 5999 ->1 6000 ->1
[85] 6110 ->1 6155 ->1 6221 ->1 6244 ->1 6266 ->1 6355 ->1 6443 ->1 6554 ->1 6665 ->1 6711 ->1 6777 ->1 6789 ->1
+ ... omitted several edges
> plot(newG70)
```



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

### Alpha Centrality

Using igraph's **alpha\_centrality()** function, we attempt to determine the alpha centrality of our subgraph. Even after hours of operation, the alpha centrality on our network did not produce a result due to excessive processing costs. In terms of the alpha centrality, we observe the following outcome:

```
> # Alpha Centrality
> acsg <- alpha.centrality(newG70)
> sort(acsg,decreasing = TRUE)
   34309      2679      40221      1378      52553      10737      73317      29764      48720      58872      41842
19.933240 19.805783 19.326839 18.479848 18.441035 18.262928 18.022887 17.692362 17.666506 17.542556 17.533378
   4056      66250      74398      47132      13888      73678      74552      38687      51409      32164      10622
16.836112 16.790174 16.763320 16.758678 16.717930 16.321660 16.250039 16.236249 15.984612 15.926812 15.876868
   33308      64128       72      7918      68928      23444      6448      27753      31287      27910      53831
15.862836 15.840416 15.776413 15.542957 15.513030 15.469949 15.371618 15.163816 15.147805 15.123024 14.985749
   35420      68827      5700      56818      74861      45143      2613      5328      39676      7566      33432
14.737541 14.644284 14.551796 14.420710 14.415411 14.328322 14.120628 14.034627 13.966719 13.801855 13.731484
   33666      74384      34987      72871      36999      39665      71705      9873      49187      2841      13366
13.728975 13.725893 13.718125 13.664235 13.645679 13.607389 13.587286 13.534688 13.512505 13.425873 13.424558
   75092      16554      8731      37110      33086      11507      7381      63340      6521      21276      1644
13.312278 13.290914 13.214822 13.135566 13.086692 13.038966 13.028398 13.021705 13.010355 12.979723 12.958889
   13699      17465      5016      14688      7477      2256      1401      45932      74968      17254      57018
12.927484 12.924017 12.886971 12.800665 12.730273 12.581008 12.446029 12.435245 12.378461 12.343220 12.337927
   5640      74481      40143      16121      6443      27642      62528      74540      8243      2503      12721
12.315058 12.184325 12.175136 12.111066 12.087931 12.017504 12.017202 12.015282 11.970865 11.882887 11.871932
   37843      7122      12244      17132      20665      34545      21520      7244      35287      73956      8887
11.836337 11.819320 11.774921 11.750415 11.692184 11.675305 11.543765 11.493558 11.427344 11.399147 11.384657
   16721      23109      4001      34543      44319      34365      60861      3103      74144      71439      5388
11.374482 11.372076 11.352023 11.292017 11.222914 11.191917 11.157324 11.155687 11.152290 11.139242 11.026298
   55453      62105      48476      75483      62817      73934      64939      22277      22765      12057      64039
10.990973 10.972101 10.958767 10.903650 10.899677 10.871018 10.856034 10.850393 10.847522 10.844974 10.789911
   45976      6016      67184      453      12628      13320      66106      74409      73830      23788      265
10.667539 10.654416 10.563463 10.553771 10.544720 10.540266 10.498367 10.494934 10.492117 10.477339 10.474891
   13577      2507      68198      10825      12567      30220      752      6441      62584      1072      73580
10.462288 10.455232 10.433090 10.428644 10.419034 10.399399 10.396477 10.375565 10.372516 10.366785 10.327756
   72361      74544      73793      24577      74475      58784      74411      61795      548      1321      11248
10.285911 10.285487 10.258719 10.253054 10.252084 10.251565 10.244806 10.236382 10.201818 10.197988 10.188328
   4.036109  4.035629  4.034951  4.029745  4.022428  4.019221  4.017913  4.017268  4.015592  4.004132  4.000832
   5965      52220      23998      7000      73835      37998      19909      74389      67440      61140      20642
3.999352  3.996883  3.996679  3.995185  3.989776  3.986705  3.982851  3.974237  3.965520  3.964734  3.956667
   15943      2917      57495      59828      38442      11724      2346      31909      15921      13788      24443
3.953222  3.950676  3.947066  3.945885  3.941418  3.938078  3.926752  3.926611  3.923872  3.919631  3.917580
   21097      31986      48065      3336      2889      4931      1292      39143      27542      14932      74516
3.910663  3.910039  3.908333  3.896985  3.896952  3.884783  3.876262  3.875065  3.862349  3.843082  3.838572
   74447      64383      1864      44075      75133      7622      24455      13801      14087      29698      2906
3.835325  3.832287  3.824540  3.820029  3.819836  3.819125  3.811162  3.804250  3.794464  3.788160  3.770851
   1939      769      34754      25354      29376      617      75621      43920      18943      1375      24732
3.763158  3.759452  3.756282  3.746919  3.744317  3.744097  3.744066  3.742642  3.734205  3.726825  3.724722
   50953      22998      5242      12366      46653      17265      22477      33275      27110      30886      13832
3.724048  3.705560  3.703926  3.698005  3.697677  3.697403  3.696013  3.687579  3.684224  3.682320  3.681366
   5389      17520      58439      74017      74915      10154      13399      23054      1934      73863      2557
3.675435  3.672976  3.662318  3.659418  3.654827  3.646382  3.644172  3.643856  3.640008  3.639026  3.637145
   64573      30664      11943      3380      31997      34666      32476      11594      20110      25287
3.636042  3.633471  3.632161  3.627822  3.617851  3.615803  3.615740      3.615636  3.612112  3.609968
[ reached getOption("max.print") -- omitted 3322 entries ]
```

## Central Node

We can find the central node in two ways,

First, based on the sum of in and out degree.

```
> # Central Node
> central_node <- which.max(degree(g, mode = "all"))
> central_node
8886
156
```

Second, based on the node with the highest betweenness

```
> bet <- betweenness(g)
> central_node <- which.max(bet)
> central_node
8886
156
```

## Longest Path

We use the **components()** function to extract the longest path, which must be in the greatest connected component. We then construct a subgraph with vertices inside that largest connected component. We also give each vertex attribute a degree.

Next, we can calculate the greatest distance from a single vertex using DFS.

```
> # Longest Path
> sg = induced.subgraph(g, which(components(g)$membership == 1))
Warning message:
`induced.subgraph()` was deprecated in igraph 2.0.0.
i Please use `induced_subgraph()` instead.
This warning is displayed once every 8 hours.
Call `lifecycle::last_lifecycle_warnings()` to see
where this warning was generated.
> V(sg)$degree = degree(sg)
> result = dfs(sg, root = 1, dist = TRUE)$dist
> sort(result, decreasing = TRUE)
61710 67561 36935 61709 61711 63401 63893 38748 69949
 9955 9955 9954 9954 9954 9954 9954 9953 9953
57193 61679 61680 61681 61682 41214 41212 41213 32077
 9953 9953 9953 9953 9953 9953 9953 9953 9952
75287 24951 55548 57192 59268 59269 59582 59583 60676
 9952 9952 9952 9952 9952 9952 9952 9952 9952
60677 68358 68359 61180 61687 61688 67355 4166 9044
 9952 9952 9952 9952 9952 9952 9951 9951 9951
32087 49257 24360 30164 55547 32139 39283 57788 57789
 9951 9951 9951 9951 9951 9951 9951 9951 9951
28171 59581 61099 28673 62459 67773 30931 13882 25170
 9951 9951 9951 9951 9951 9950 9950 9950 9950
46825 32078 32079 25244 15469 54098 18327 55830 55831
```

```

17257 74395 43558 15942 44629 40125 65806 24995 9705
9926 9926 9926 9926 9926 9926 9926 9926 9926
34314 73720 15532 49896 49897 50160 50682 25764 41294
9926 9926 9926 9926 9926 9926 9926 9926 9926
50697 50699 53932 38680 57888 57889 58316 58318 58425
9926 9926 9926 9926 9926 9926 9926 9926 9926
21792 37351 61176 61177 37100 39607 39612 65386 65387
9926 9926 9926 9926 9926 9926 9926 9926 9926
65388 65389 65390 37101 37102 39203 39601 39602 70975
9926 9926 9926 9926 9926 9926 9926 9926 9925
7750
9925
[ reached getOption("max.print") -- omitted 74877 entries ]

```

## Largest Clique

We get the minimal degree needed for every vertex in the greatest clique, as well as the largest cliques in our graph g directly using the **largest\_cliques()** method. To find the greatest cliques, we utilize the binary search approach and the splitting graph.

```

> # Largest Clique
> largest_cliques(g)
[[1]]
+ 23/75879 vertices, named, from b69a5b5:
[1] 26109 45553 75103 38109 226 44441 68882 15553 337 22220 49998 56661 2111 2556 74770 31442 2222 1111 50109
[20] 38775 75547 17775 21108

[[2]]
+ 23/75879 vertices, named, from b69a5b5:
[1] 26109 45553 75103 38109 226 44441 68882 15553 337 22220 14776 56661 75436 1889 2556 17775 38775 75547 21108
[20] 50109 59883 74770 62661

[[3]]
+ 23/75879 vertices, named, from b69a5b5:
[1] 26109 45553 75103 38109 226 44441 68882 15553 337 22220 14776 56661 75436 1889 2556 17775 38775 75547 21108
[20] 50109 59883 74770 2222

[[4]]
+ 23/75879 vertices, named, from b69a5b5:
[1] 26109 45553 33443 68882 38109 226 44441 2222 337 22220 15553 56661 2111 31442 1111 17775 49998 50109 21108
[20] 74770 75547 2556 38775

[[5]]
+ 23/75879 vertices, named, from b69a5b5:
[1] 26109 45553 33443 68882 38109 226 44441 2222 337 22220 15553 56661 1889 14776 75436 38775 21108 17775 50109
[20] 2556 75547 74770 59883

```

```

[[6]]
+ 23/75879 vertices, named, from b69a5b5:
[1] 26109 45553 33443 68882 38109 27775 74770 2556 44441 2222 31442 50109 337 22220 15553 17775 56661 2111 21108
[20] 38775 1111 75547 49998

[[7]]
+ 23/75879 vertices, named, from b69a5b5:
[1] 71104 45553 75103 38109 68882 226 44441 448 15553 337 22220 14776 1889 75436 7221 2222 44442 69993 19997
[20] 62550 1778 33554 49997

[[8]]
+ 23/75879 vertices, named, from b69a5b5:
[1] 71104 45553 75103 38109 68882 226 44441 56661 15553 337 33554 2222 44442 22220 62550 49997 14776 1889 7221
[20] 1778 69993 19997 75436

[[9]]
+ 23/75879 vertices, named, from b69a5b5:
[1] 71104 45553 33443 68882 38109 337 2222 226 448 22220 14776 44441 15553 1889 7221 62550 75436 69993 19997
[20] 44442 1778 33554 49997

[[10]]
+ 23/75879 vertices, named, from b69a5b5:
[1] 71104 45553 33443 68882 38109 337 2222 226 56661 15553 44441 33554 22220 14776 1889 42219 62550 7221 75436
[20] 69993 49997 44442 1778

[[11]]
+ 23/75879 vertices, named, from b69a5b5:
[1] 71104 45553 33443 68882 38109 337 2222 226 56661 15553 44441 33554 22220 14776 1889 42219 62550 7221 75436
[20] 69993 49997 44442 17775

[[12]]
+ 23/75879 vertices, named, from b69a5b5:
[1] 71104 45553 33443 68882 38109 337 2222 226 56661 15553 44441 33554 22220 14776 1889 19997 1778 62550 69993
[20] 7221 75436 49997 44442

[[13]]
+ 23/75879 vertices, named, from b69a5b5:
[1] 71104 44441 75103 226 38109 18886 68882 448 2222 1889 337 22220 7221 15553 44442 14776 49997 62550 33554
[20] 1778 19997 69993 75436

[[14]]
+ 23/75879 vertices, named, from b69a5b5:
[1] 71104 44441 75103 226 38109 18886 68882 56661 337 15553 33554 2222 75436 1889 14776 22220 62550 49997 44442
[20] 69993 7221 1778 19997

[[15]]
+ 23/75879 vertices, named, from b69a5b5:
[1] 71104 44441 33443 18886 38109 226 68882 337 2222 22220 1889 448 75436 14776 7221 62550 69993 19997 15553
[20] 1778 33554 44442 49997

[[16]]
+ 23/75879 vertices, named, from b69a5b5:
[1] 71104 44441 33443 18886 38109 226 68882 337 2222 22220 1889 56661 15553 69993 75436 14776 33554 62550 7221
[20] 49997 44442 17775 42219

[[17]]
+ 23/75879 vertices, named, from b69a5b5:
[1] 71104 44441 33443 18886 38109 226 68882 337 2222 22220 1889 56661 15553 69993 75436 14776 33554 62550 7221
[20] 49997 44442 1778 42219

[[18]]
+ 23/75879 vertices, named, from b69a5b5:
[1] 71104 44441 33443 18886 38109 226 68882 337 2222 22220 1889 56661 15553 69993 75436 14776 33554 62550 7221
[20] 49997 44442 1778 19997

```

## Ego(s)

We calculated the egos for all the vertices as shown. In our result, most of the values get omitted.

```
[55] 55086 66416 75784 75835 35121 25732 54820 20554 54931 13677 13965 8198 33743 40964 13666 61750 75126 74148
[73] 73311 10021 75831 45443 38376 64327 74281 9953 6833 11111 13999 37221 44999 75764 593 75852 65216 60506
[91] 10254 73299 75767 75860 33532 53131 6822 63539 75773 75810 74981 75857 54020 17465 20187 75854 16143 73693
[109] 73499 44987 75842 991 75771 75859 13011 75845 75765 59007 30021 75794 36631 75782 75818 75749 75844 73808
[127] 75849 75766 70550 60883 72005 75841 75371 75850 75768 75809 75824 27698 75856 75798 75861 75780 73757 75853
[145] 75811 75756 16209 75796 75512 75761 21347 75760 22274 75808 75828 75802 75375 30752 67501 69241 69266 71914
[163] 71915 71917 71918 71920 74758 75750 75751 75752 75753 75754 75755 75757 75759 75762 75763 75772 75774 75775
+ ... omitted several vertices

[[997]]
+ 35/75879 vertices, named, from b69a5b5:
 [1] 32331 4222 5000 8886 13554 19886 20886 23331 33330 33776 44441 63327 73658 326 24220 30442 34554 37998 45331
[20] 2 75647 55552 4457 93 70993 1734 66549 16043 1434 73392 73614 75571 1401 74118 48431 24265 66516 9887
[39] 75059 60 24466 20809 28498 72771 1512 73213 74314 73642 28409 75634 74355 40243 75862

[[998]]
+ 35/75879 vertices, named, from b69a5b5:
 [1] 32442 7800 8886 12187 12788 15331 15921 20220 58883 16265 16332 25442 70549 2 2112 35409 47365 25398 11743
[20] 35132 12444 74636 9953 7599 31409 36221 43942 62806 75176 74615 54620 75864 26987 45599 75863

[[999]]
+ 978/75879 vertices, named, from b69a5b5:
 [1] 32487 1334 2778 2889 3111 3145 3334 4778 5778 6855 7665 12388 13065 14276 19997 21109 21553 22332
[19] 31664 32576 36553 37775 42031 43330 47876 48886 49886 54441 59994 61106 67051 67106 73281 74103 83 326
[37] 354 939 1337 1375 1589 1750 2301 4211 4531 4801 4890 5001 5056 5145 5278 5655 5678 6889
[55] 6955 9431 10132 10965 12987 18376 18932 24154 24443 27331 27799 31920 35554 36887 36998 39409 44197 44330
[73] 47121 49553 54498 55264 55331 57473 70039 70549 70882 71305 72550 73249 74212 74443 74747 74867 74925 2
[91] 2056 2112 74870 3501 9054 15209 17854 65883 75581 2232 11888 14643 55802 61784 74970 75847 1952 28032
[109] 29498 75479 7954 37954 1689 33943 35221 75138 2013 73485 16043 60628 69105 74735 2608 7088 7099 7458
[127] 7460 7462 9243 55086 73326 71038 18032 29831 8343 15566 21731 27487 54875 9043 74993 4545 34232 61262
[145] 567 31742 46843 55242 227 3200 4119 7108 8076 9387 14329 14447 15773 18365 18720 30032 34699 35365
[163] 44264 45932 54909 61240 61795 62972 63795 1386 15767 18232 53653 68771 74725 38864 7508 1975 4756 6281
+ ... omitted several vertices

[[1000]]
+ 24/75879 vertices, named, from b69a5b5:
 [1] 32553 448 1889 2222 5000 7221 9442 13331 26664 44442 60550 2 6621 238 3845 73270 3923 18132 3335
[20] 3579 44364 19631 31042 75865

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

## Power Centrality

Because it is less computationally demanding than operating on the original graph, we run the **power\_centrality()** function on the subgraph and obtain the following outcome:

```

> # Power Centrality
> pc <- power_centrality(sg, exponent = 0.8)
> sort(pc,decreasing = TRUE)
  50898     74398     18943     2557     17065
3.8065708 3.4263058 3.3555889 3.2372445 3.1841918
  30565     43975     11865     23309     75092
3.0756045 3.0654297 3.0593137 3.0570830 3.0474227
  7778      1975     6077     75231     55555
2.9999894 2.9886769 2.8507025 2.7802631 2.7800905
  7949      13788     73727     9873     1601
2.7579463 2.7408055 2.7396547 2.7308880 2.7045923
  11952     10013     47154     33799     12965
2.6992216 2.6592313 2.6024564 2.5967170 2.5747038
  73934     75593     2668     27520     72071
2.5589962 2.5291807 2.5269885 2.4998050 2.4982015
  2256      76      39465     68705     59872
2.4620935 2.4546216 2.4522640 2.4411813 2.4212111
  5055      74739     50465     1044     14566
2.4184768 2.4113449 2.3999751 2.3849355 2.3450448
  0.5158922 0.5157707 0.5151705 0.5145553 0.5124539
  4667      7388     10737     55153     23499
0.5118930 0.5116844 0.5111054 0.5105826 0.5099342
  64772     73684     61428     6033     1334
0.5075373 0.5073847 0.5072200 0.5062557 0.5051748
  4955      52864     75139     73770     58661
0.5050106 0.4975121 0.4967682 0.4957897 0.4947362
  14887     6213     66250     10432     45710
0.4943133 0.4943075 0.4939097 0.4937603 0.4932935
  62484     24087     34420     74231     3083
0.4928185 0.4920372 0.4884251 0.4874478 0.4873675
  43764     73517     25976     3244     66884
0.4861649 0.4851846 0.4846476 0.4843152 0.4840420
  7443      10646     74559     5711     4568
0.4834032 0.4830957 0.4821473 0.4820341 0.4798337
[ reached getOption("max.print") -- omitted 3322 entries ]

```

## Discussion

Our understanding of working with datasets and creating graphs from them—especially medium-to large-scale datasets—is improved by this study. We gained practical experience in building our own functions, using the igraph and sna packages, and utilizing R. We improved our abilities to plot and visualize graphs, simplify them, and change their parameters to make them easier to read. We obtain important insights into the structure and relationships inside the graph using the numerous graph analytics functions that are described in this project.

We gained knowledge of key metrics for comprehending the issue space, including power centrality, greatest cliques, egos, longest paths, and alpha centrality. We also studied how random walks can be used to locate communities in a graph. We can find patterns, structures, and links in the data and create efficient methods for examining and visualizing big, intricate networks by comprehending these metrics.

After completing this project, we can confidently work with R to generate and plot graphs, perform graph analytics on large datasets, apply various functions and measures/metrics to the graph that allow us to draw different conclusions about the graph and the problem domain, and simplify graphs for better analysis.