Introduction to Big Data and Analytics

Group 4 - Project 1

Point #1

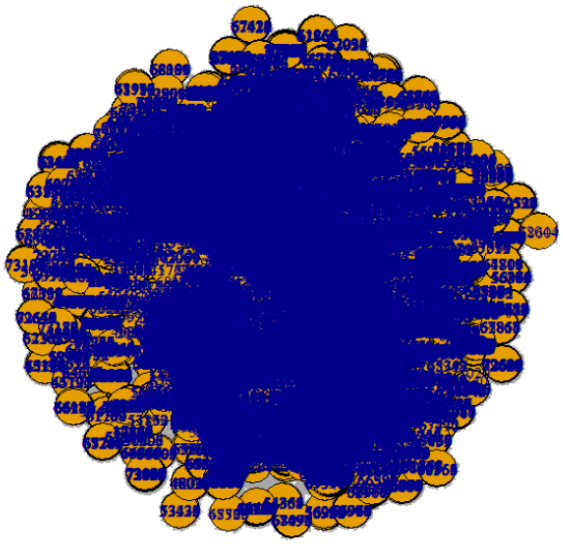
Data Set - soc-Epinions1_adj.tsv (from blackboard)
We are using data set from this file which is available on blackboard. The file is stored in the local system under the path -
The format of each row in the data set is node1, node2, #edges. Since the number of edges for each row is 1, this column is removed from the data set while doing the computations.

Point #2

To plot the initial data set graph, we are using the **igraph** package.
Steps for plotting -

1. Read the data from the tsv file stored on the local system.
2. Delete the column number of edges since its value is 1 for every row.
3. Convert the data to matrix format.
4. Create vectors v1 and v2 for columns 1 and 2 respectively.
5. Use the inbuilt function to create a graph data object and plot it.

Plotted graph -

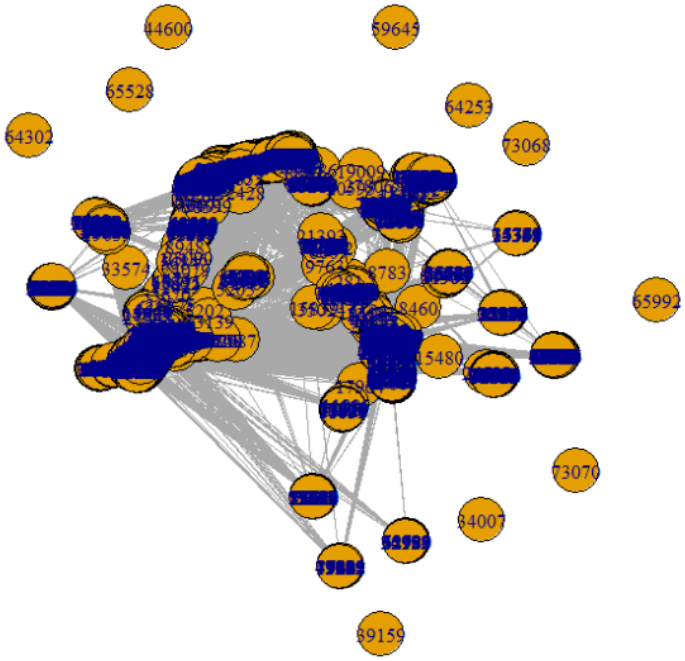


Point #3

The original data set is medium-sized (10 million rows) and hence it needs to be simplified before performing operations given in the rubric to make the graph readable when its plotted.
The steps for simplifying the graph are -

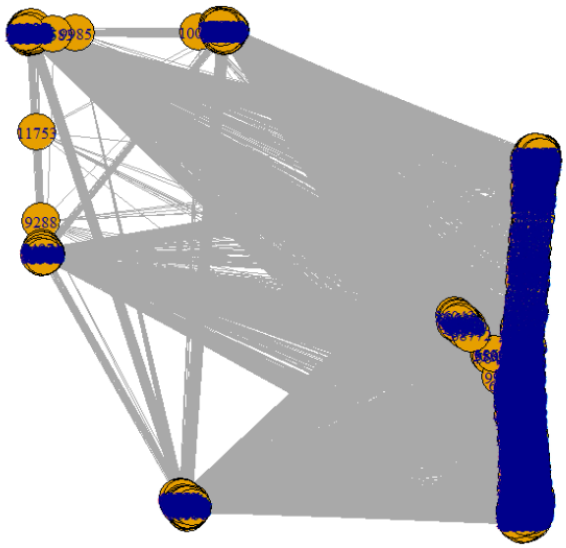
Version 1

We create an undirected graph data object so the arrows are not plotted to make the graph more visible.
First, we use the inbuilt function **simplify** given by **igraph** package which simplifies the graph by removing any self-loops and also removing multiple edges between two edges to a single edge.
After that, we delete the vertices whose degree is less than the average degree of the whole graph.
This graph is better than the original graph but still, it's not so visible and requires further simplification.



Version 2

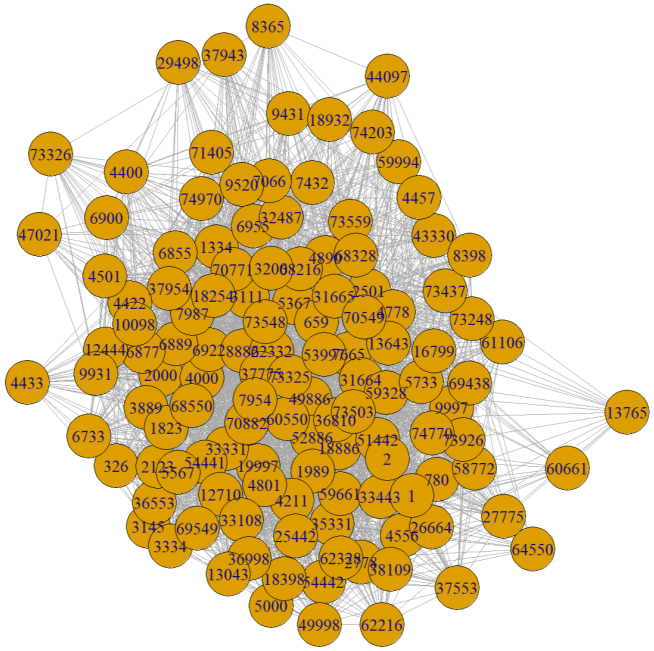
The second attempt for simplification was to calculate the average coreness of the graph. The vertices whose coreness was less than average coreness are deleted. The resultant graph had fewer nodes compared to the previous version but it was also not so readable.



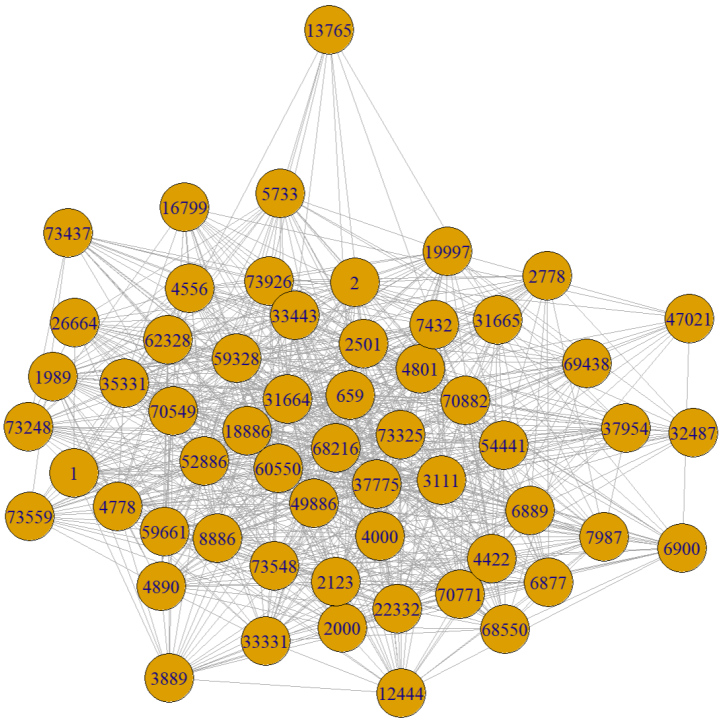
Version 3

The third and final attempt for simplification involved calculating the betweenness and closeness of nodes. The top 100 and 50 such nodes are selected and a graph is plotted for each. Both these graphs are much better than the previous results. Finally, we went ahead with selecting the top 50 nodes because it made the graph more readable and at the same time made sure we are not choosing very few nodes (like 15 or 25). This is a very small sample of the whole data and cannot be a good representation of the initial data set but it will give us a good idea to get an overall picture of the data set.

Top 100 nodes -



Top 50 nodes -



Results of the functions given in the rubric -

- Vertices

```
> # get vertices of the graph
> V(simplified_g)
+ 57/57 vertices, named, from 918324b:
 [1] 2000  2778  3111  3889  4000  4556  4778  6922  8886  18886 19997 26664 31664
[14] 33331 35331 37775 49886 52886 54441 59328 59661 60550 62328 73248 73325 73926
[27] 659   1989  4890  5733  31665 32487 33443 36998 49998 51442 68216 69438 70549
[40] 70882 1     2     2123  70771 6877  7987  6900  73559 37954 4422  73548 73437
[53] 4433  64550 16799 69549 68550
```

- Edges

```
> # get edges of the graph
> E(simplified_g)
+ 1684/1684 edges from 918324b (vertex names):
 [1] 2000->3889  2000->8886  2000->31664 2000->35331 2000->37775 2000->49886
 [7] 2000->52886 2000->54441 2000->59328 2000->59661 2000->60550 2000->73325
[13] 2000->4890  2000->31665 2000->36998 2000->51442 2000->70549 2000->1
[19] 2000->2     2000->2123  2000->70771 2000->6877  2000->7987  2000->6900
[25] 2000->73559 2000->4422  2000->73548 2000->4433  2000->69549 2000->68550
[31] 2778->4556  2778->8886  2778->18886 2778->31664 2778->37775 2778->54441
[37] 2778->59328 2778->60550 2778->62328 2778->73926 2778->659   2778->31665
[43] 2778->32487 2778->51442 2778->70882 2778->1     2778->2123  2778->6877
[49] 2778->7987  3111->4000  3111->4778  3111->6922  3111->8886  3111->18886
[55] 3111->19997 3111->31664 3111->33331 3111->35331 3111->37775 3111->49886
+ ... omitted several edges
```

- Adjacency matrix

```
> # get adjacency matrix of the graph
> get.adjacency(simplified_g)
57 x 57 sparse Matrix of class "dgCMatrix"
   [[ suppressing 36 column names ‘2000’, ‘2778’, ‘3111’ ... ]]
   [[ suppressing 36 column names ‘2000’, ‘2778’, ‘3111’ ... ]]

2000  . . . 1 . . . . 1 . . . 1 . 1 1 1 1 1 1 1 1 . . 1 . . . 1 . 1 . . 1 . 1 ......
2778  . . . . . 1 . . 1 1 . . 1 . . 1 . . 1 1 . 1 1 . . 1 1 . . . 1 1 . . . 1 ......
3111  . . . . 1 . 1 1 1 1 1 . 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . . 1 1 1 . . . 1 ......
3889  1 . . . 1 . 1 1 . 1 . . 1 1 . 1 1 1 . . . 1 1 . . . 1 1 1 . . . . 1 1 . ......
4000  . . 1 1 . 1 . 1 1 1 . . 1 . . 1 1 . 1 . . 1 . 1 1 1 1 . 1 . . . 1 . 1 . ......
4556  . 1 . . 1 . 1 1 1 1 1 1 1 . 1 . 1 . 1 1 . 1 1 1 . 1 . 1 . 1 . . 1 1 1 1 ......
4778  . . 1 1 . 1 . . 1 1 . . 1 1 1 . 1 1 . 1 1 1 1 . 1 1 1 1 1 . . 1 1 . . . ......
6922  . . 1 1 1 1 . . 1 . 1 . . . . 1 . 1 . 1 . 1 1 . . 1 . . 1 . . . . . . 1 ......
8886  1 1 1 . 1 1 1 1 . 1 . . 1 . 1 1 1 1 1 1 1 1 1 . 1 1 1 . 1 . . . . . . 1 ......
18886 . 1 1 1 1 1 1 . 1 . 1 1 1 1 1 1 1 1 1 1 1 1 . 1 1 1 1 1 . . . . 1 1 1 1 ......
19997 . . 1 . . 1 . 1 . 1 . . 1 1 . 1 . 1 1 1 . 1 1 . 1 1 . . . 1 1 1 1 1 1 ......
26664 . . . . . 1 . . . 1 . . 1 . 1 1 1 1 . 1 1 1 . 1 1 . 1 . . 1 . . 1 . . 1 ......
31664 1 1 1 1 1 1 1 . 1 1 1 1 . . 1 1 1 1 . 1 . 1 1 1 . 1 1 1 1 1 . 1 1 . 1 1 ......
33331 . . 1 1 . . 1 . . 1 1 . . . . 1 . 1 . . . 1 . . 1 . 1 1 1 . . . . 1 1 1 ......

 ..............................
 ........suppressing 21 columns and 30 rows in show(); maybe adjust 'options(max.print=
*, width = *)'
 ..............................
   [[ suppressing 36 column names ‘2000’, ‘2778’, ‘3111’ ... ]]

6877  1 1 1 1 . 1 . . . . . . 1 . . 1 1 . 1 . 1 1 1 . . . 1 . 1 . 1 . . 1 . . ......
7987  1 1 1 . 1 . . . . 1 . . . 1 . 1 1 1 . . . 1 . . 1 1 1 . . . 1 . . . . 1 ......
6900  1 . 1 . . . . 1 . . . . . . . 1 1 1 1 . . . . . . 1 . . . . 1 . . . . . ......
73559 1 . . . 1 1 . 1 1 1 . . 1 . . 1 1 1 . 1 . 1 1 . . 1 1 . 1 . . . . . . 1 ......
37954 . . 1 . . . 1 . . . 1 . 1 . . 1 1 1 1 1 . 1 . . 1 . 1 . . . 1 1 . . . . ......
4422  1 . 1 1 1 1 . 1 1 . 1 . 1 . . 1 1 . 1 1 . 1 1 . . . . . 1 . 1 . . . . . ......
73548 1 . 1 . 1 . 1 . 1 . . 1 . 1 1 1 1 1 1 1 1 1 1 . 1 1 1 1 1 . 1 . . . 1 1 ......
73437 . . . . . 1 . . . 1 . . 1 . . 1 1 1 . 1 . 1 1 1 . 1 1 . . . 1 . . . . 1 ......
4433  1 . . 1 1 . . . 1 1 . . . . . 1 . . 1 . . 1 . . . . . . . . . 1 . . . . ......
64550 . . . . . 1 . . . 1 . . 1 . 1 . 1 . . . 1 1 1 1 . 1 . . . . . 1 1 . . 1 ......
16799 . . . . . 1 . . . 1 . . 1 . 1 1 . 1 . 1 1 1 . 1 . 1 1 . 1 . . . 1 . . 1 ......
69549 1 . 1 1 1 1 . 1 1 1 . . 1 1 1 1 1 1 1 1 1 . 1 . . 1 . 1 . . . . . 1 . . 1 ......
68550 1 . 1 . 1 . . 1 . 1 . 1 . 1 . 1 1 1 1 1 1 1 1 . . . . 1 . . . . . 1 . 1 1 ......
```

- Density

```
> #get density of the graph
> graph.density(simplified_g)
[1] 0.5275689
```

- Edge density

```
> # get edge density
> edge_density(simplified_g)
[1] 0.5275689
> # get edge density with loops = T
> edge_density(simplified_g, loops = T)
[1] 0.5183133
```

- Degree

```
> # get the degree of each node in the graph
> igraph::degree(simplified_g)
 2000  2778  3111  3889  4000  4556  4778  6922  8886 18886 19997
   60    38    80    50    60    70    54    50    66    86    56
26664 31664 33331 35331 37775 49886 52886 54441 59328 59661 60550
   48    82    52    64    92    88    80    76    76    60   106
62328 73248 73325 73926   659  1989  4890  5733 31665 32487 33443
   60    44    74    68    68    50    58    48    56    30    66
36998 49998 51442 68216 69438 70549 70882     1     2  2123 70771
   56    52    80    66    42    70    84    52    60    66    62
 6877  7987  6900 73559 37954  4422 73548 73437  4433 64550 16799
   50    52    32    40    36    54    66    32    28    26    34
69549 68550
   56    56
```

- Betweenness centrality

```
> # get betweenness centrality of the graph
> igraph::centr_betw(simplified_g)
$res
 [1]  23.841741   7.861101  44.714012  15.955809  22.220026
 [6]  39.565579  16.243248  13.930931  21.680952  62.505726
[11]  19.702098   8.062623  62.974504  14.065041  23.231503
[16]  79.200214  70.181314  50.807215  47.869718  46.549941
[21]  26.205536 103.088886  26.125392  12.531432  29.706110
[26]  41.499494  37.453957  10.081011  23.929164  10.038695
[31]  26.443215   6.957652  24.794121  20.834684  11.273298
[36]  52.152226  39.894973   6.033886  31.250519  58.404329
[41]  14.199052  21.021170  32.180188  26.609649  12.096865
[46]  13.538806   2.818812   7.723437   5.427177  14.203909
[51]  25.414571   4.117536   2.797681   3.313006   3.963371
[56]  12.628106  16.084788

$centralization
[1] 0.02532506

$theoretical_max
[1] 172480
```

- Closeness centrality

```
> # get closeness centrality of a node in the graph
> igraph::centr_clo(simplified_g)
$res
 [1] 0.6829268 0.6021505 0.7777778 0.6436782 0.6829268 0.7272727
 [7] 0.6588235 0.6436782 0.7088608 0.8115942 0.6666667 0.6363636
[13] 0.7887324 0.6511628 0.7000000 0.8484848 0.8235294 0.7777778
[19] 0.7567568 0.7567568 0.6829268 0.9491525 0.6829268 0.6222222
[25] 0.7466667 0.7179487 0.7179487 0.6436782 0.6746988 0.6363636
[31] 0.6666667 0.5773196 0.7088608 0.6666667 0.6511628 0.7777778
[37] 0.7088608 0.6153846 0.7272727 0.8000000 0.6511628 0.6829268
[43] 0.7088608 0.6913580 0.6436782 0.6511628 0.5833333 0.6086957
[49] 0.5957447 0.6588235 0.7088608 0.5833333 0.5714286 0.5656566
[55] 0.5894737 0.6666667 0.6666667

$centralization
[1] 0.271794

$theoretical_max
[1] 55.01754
```

- Alpha centrality

```
> alpha_centrality(simplified_g)
        2000         2778         3111         3889         4000
 0.183757269 -0.130466378 -0.151811318 -0.052420650 -0.014378932
        4556         4778         6922         8886        18886
-0.086205398 -0.233025207 -0.041382426 -0.134668355 -0.330945398
       19997        26664        31664        33331        35331
 0.023531367  0.315779614  0.004607289  0.020470729  0.209215224
       37775        49886        52886        54441        59328
-0.052311844 -0.170429457 -0.165877183 -0.102995557 -0.041641090
       59661        60550        62328        73248        73325
-0.277000729 -0.143851691  0.041944285  0.216596020  0.266218019
       73926          659         1989         4890         5733
-0.046896040  0.017115598 -0.044821604  0.069041801  0.168084364
       31665        32487        33443        36998        49998
-0.208787760  0.007602821 -0.125086553 -0.125104907  0.084382821
       51442        68216        69438        70549        70882
-0.094474902 -0.136364023 -0.020310132 -0.286573177 -0.116644692
           1            2         2123        70771         6877
 0.105812166  0.080796060  0.194195494 -0.008934931 -0.146703904
        7987         6900        73559        37954         4422
 0.004382600  0.146009479  0.044117201 -0.006086632  0.079759326
       73548        73437         4433        64550        16799
 0.234561089 -0.153404392 -0.030443023  0.205075472 -0.100723610
       69549        68550
-0.058723745 -0.154015575
```

- Shortest path

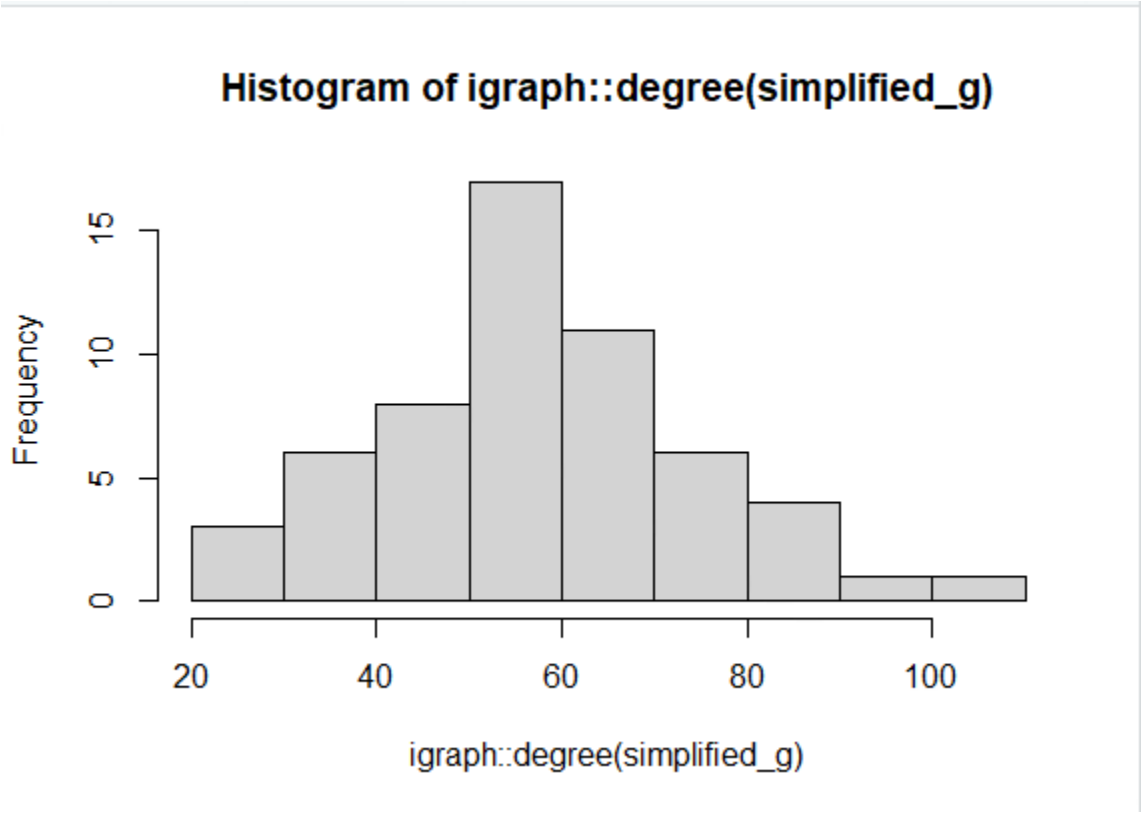
```
> # get the shortest path between two nodes
> igraph::shortest.paths(simplified_g)
      2000 2778 3111 3889 4000 4556 4778 6922 8886 18886 19997
2000     0    2    2    1    2    2    2    2    1     2     2
2778     2    0    2    2    2    1    2    2    1     1     2
3111     2    2    0    2    1    2    1    1    1     1     1
3889     1    2    2    0    1    2    1    1    2     1     2
4000     2    2    1    1    0    1    2    1    1     1     2
4556     2    1    2    2    1    0    1    1    1     1     1
4778     2    2    1    1    2    1    0    2    1     1     2
6922     2    2    1    1    1    1    2    0    1     2     1
8886     1    1    1    2    1    1    1    1    0     1     2
18886    2    1    1    1    1    1    1    2    1     0     1
19997    2    2    1    2    2    1    2    1    2     1     0
26664    2    2    2    2    2    1    2    2    2     1     2
31664    1    1    1    1    1    1    1    2    1     1     1
33331    2    2    1    1    2    2    1    2    2     1     1
35331    1    2    1    2    2    1    1    2    1     1     2
37775    1    1    1    1    1    2    2    1    1     1     1
49886    1    2    1    1    1    1    1    2    1     1     2
      26664 31664 33331 35331 37775 49886 52886 54441 59328 59661
2000      2     1     2     1     1     1     1     1     1     1
2778      2     1     2     2     1     2     2     1     1     2
3111      2     1     1     1     1     1     1     1     1     1
3889      2     1     1     2     1     1     1     2     2     2
4000      2     1     2     2     1     1     2     1     2     2
4556      1     1     2     1     2     1     2     1     1     2
4778      2     1     1     1     2     1     1     2     1     1
6922      2     2     2     2     1     2     1     2     1     2
8886      2     1     2     1     1     1     1     1     1     1
18886     1     1     1     1     1     1     1     1     1     1
19997     2     1     1     2     1     2     1     1     1     2
26664     0     1     2     1     1     1     1     2     1     1
31664     1     0     2     1     1     1     1     2     1     2
33331     2     2     0     2     1     2     1     2     2     2
35331     1     1     2     0     1     2     1     1     1     2
37775     1     1     1     1     0     1     1     1     1     1
49886     1     1     2     2     1     0     1     1     1     1
```

- Actual shortest path

```
> # get actual shortest paths between two nodes
> shortest_path <- shortest_paths(simplified_g, from = 1, to = 5)
> shortest_path$vpath[[1]]
+ 3/57 vertices, named, from 918324b:
[1] 2000 3889 4000
```

- Histogram of degree

```
> # get histogram of the degree of nodes in the graph
> hist(igraph::degree(simplified_g))
```



- Edge density

```
> # get edge density
> edge_density(simplified_g)
[1] 0.5275689
> # get edge density with loops = T
> edge_density(simplified_g, loops = T)
[1] 0.5183133
```

- Diameter

```
> # get diameter of the graph
> igraph::diameter(simplified_g)
[1] 2
```

- Max cliques

```
> # get max cliques for vertex 1
> max_cliques(simplified_g, min = NULL, max = NULL)[[1]]
+ 6/57 vertices, named, from 918324b:
[1] 64550 59661 33443 60550 49886 18886
```

- Size of the largest clique

```
> # find the largest cliques in the graph
> all_cliques <- cliques(simplified_g)
Warning message:
In cliques(simplified_g) :
  At core/cliques/cliquer_wrapper.c:57 : Edge directions are ignored f
or clique calculations.
> max_size <- max(sapply(all_cliques, length))
> max_size
[1] 12
```

The density of a graph is the measure of how well connected the graph is. In our case, the density value is around 0.5 which illustrates that the simplified graph is well-connected and the opinion reviews for this data set are highly valued.

Point #4

Following operations are performed and the values are determined -

- Central nodes

```
> # determining central nodes
> # calculate degree centrality
> degree_centrality <- degree(simplified_g)
> # calculate betweenness centrality
> betweenness_centrality <- betweenness(simplified_g)
> # calculate eigenvector centrality
> eigenvector_centrality <- eigen_centrality(simplified_g)$vector
> central_nodes <- which(degree_centrality == max(degree_centrality) &
betweenness_centrality == max(betweenness_centrality) & eigenvector_ce
ntrality == max(eigenvector_centrality))
> print(central_nodes)
60550
   22
```

- Longest path

```
> # determine longest path of the graph
> diameter <- get_diameter(simplified_g)
> diameter
+ 3/57 vertices, named, from 918324b:
[1] 2000 3889 4000
```

- Largest cliques

```
> # determine largest cliques of the graph
> all_cliques <- cliques(simplified_g)
Warning message:
In cliques(simplified_g) :
  At core/cliques/cliquer_wrapper.c:57 : Edge directions are ignored f
or clique calculations.
> largest_size <- max(sapply(all_cliques, length))
> largest_cliques <- all_cliques[sapply(all_cliques, length) == larges
t_size]
> largest_cliques
[[1]]
+ 12/57 vertices, named, from 918324b:
 [1] 3111  8886  35331 37775 52886 54441 60550 73325 51442 70549
[11] 70882 73548

[[2]]
+ 12/57 vertices, named, from 918324b:
 [1] 3111  8886  18886 35331 37775 52886 54441 60550 73325 51442
[11] 70549 70882

[[3]]
+ 12/57 vertices, named, from 918324b:
 [1] 3111  8886  35331 37775 52886 54441 60550 73325 659   70549
[11] 70882 73548

[[4]]
+ 12/57 vertices, named, from 918324b:
 [1] 3111  8886  18886 35331 37775 52886 54441 60550 73325 659
[11] 70549 70882

[[5]]
+ 12/57 vertices, named, from 918324b:
 [1] 3111  8886  37775 49886 52886 54441 60550 73325 51442 70549
[11] 70882 73548

[[6]]
+ 12/57 vertices, named, from 918324b:
 [1] 3111  8886  18886 37775 49886 52886 54441 60550 73325 51442
[11] 70549 70882
```

- Egos

```
> # determine egos of the graph
> ego(simplified_g)
[[1]]
+ 31/57 vertices, named, from 918324b:
 [1] 2000  3889  8886  31664 35331 37775 49886 52886 54441 59328
[11] 59661 60550 73325 4890  31665 36998 51442 70549 1     2
[21] 2123  70771 6877  7987  6900  73559 4422  73548 4433  69549
[31] 68550

[[2]]
+ 20/57 vertices, named, from 918324b:
 [1] 2778  4556  8886  18886 31664 37775 54441 59328 60550 62328
[11] 73926 659   31665 32487 51442 70882 1     2123  6877  7987

[[3]]
+ 41/57 vertices, named, from 918324b:
 [1] 3111  4000  4778  6922  8886  18886 19997 31664 33331 35331
[11] 37775 49886 52886 54441 59328 59661 60550 62328 73248 73325
[21] 73926 659   5733  31665 32487 51442 68216 70549 70882 1
[31] 2     2123  70771 6877  7987  6900  37954 4422  73548 69549
[41] 68550

[[4]]
+ 26/57 vertices, named, from 918324b:
 [1] 3889  2000  4000  4778  6922  18886 31664 33331 37775 49886
[11] 52886 60550 62328 659   1989  4890  36998 49998 70549 1
[21] 2123  70771 6877  4422  4433  69549

[[5]]
+ 31/57 vertices, named, from 918324b:
 [1] 4000  3111  3889  4556  6922  8886  18886 31664 37775 49886
[11] 54441 60550 73248 73325 73926 659   4890  33443 49998 68216
[21] 70549 70882 1     2     7987  73559 4422  73548 4433  69549
[31] 68550
```

- Power centrality

```
> # determine centrality of the graph
> eigen_centrality <- eigen_centrality(simplified_g)
> eigen_centrality$vector
     2000      2778      3111      3889      4000      4556
0.6022793 0.4072237 0.7991872 0.5023216 0.6165674 0.6700522
     4778      6922      8886     18886     19997     26664
0.5599653 0.5140733 0.6921971 0.8329656 0.5708754 0.5303081
    31664     33331     35331     37775     49886     52886
0.7735944 0.5348470 0.6653541 0.8726224 0.8459300 0.7826694
    54441     59328     59661     60550     62328     73248
0.7415753 0.7360247 0.5990938 1.0000000 0.5915413 0.4452221
    73325     73926       659      1989      4890      5733
0.7617826 0.6478899 0.6626920 0.5408141 0.5723305 0.5004928
    31665     32487     33443     36998     49998     51442
0.5391351 0.3001018 0.6769784 0.5563625 0.5611091 0.7821592
    68216     69438     70549     70882         1         2
0.6134600 0.4546547 0.7155087 0.8178981 0.5454902 0.6075742
     2123     70771      6877      7987      6900     73559
0.6515320 0.6204695 0.5167932 0.5378048 0.3388932 0.4266742
    37954      4422     73548     73437      4433     64550
0.3963361 0.5548046 0.6754999 0.3539419 0.3037681 0.2785103
    16799     69549     68550
0.3832211 0.5986620 0.5801788
```

Point #5

All the deliverables are present in this report and the code is present in the file - graph_analytics.R

Point #6

We learned how to work with graphs in R and the fundamentals of graph theory through this project. We began by simplifying a graph, then deriving adjacency matrices and degrees, and finally representing graphs as matrices. This gave us a better understanding of how graphs are constructed and how their properties can be analyzed.

We also looked at different graph density, node density, and edge density measures and how they change as link factors decrease. This assisted us in

understanding how various factors can affect the overall structure and complexity of a graph.

We then investigated various centrality metrics, such as betweenness and closeness centrality. We learned how to apply these measures to real-world data sets and how they can help identify the most important nodes in a graph.

We also learned how to calculate the shortest path and number of paths between two nodes, which can aid in the discovery of hidden patterns or relationships in data. This was especially helpful in understanding the connectivity of the graph's nodes.

We also looked at cliques and learned how to find the maximum and largest cliques in a graph. We also experimented with various plotting techniques, which helped us visualize and comprehend complex data sets. Finally, we learned about alpha-centrality and ego-centrality, which are measures of the importance or influence of specific nodes in a graph. These metrics can help identify key players in a social network, for example.

Overall, we gained a better understanding of graph theory and how to apply it with R thanks to this project. We have increased our confidence in our ability to work with graphs and analyze data sets, and we believe that these skills will be useful in a variety of fields, including social network analysis, machine learning, and data analysis.