Big Data and Analytics

CS6444_80

Class project 1

By

Group - 13

Rithik Reddy P V (G23879158)
Vivek Kommareddy (G43709632)

Introduction:

Graph analytics involves analysing and extracting insights from graph-structured data, focusing on relationships between entities represented as nodes and edges. It encompasses a range of techniques and algorithms for tasks such as identifying central nodes, detecting communities, predicting links, and visualizing graph structures.

We will delve into the field of graph analytics in Class Project #1. We'll look at methods and resources for deciphering and displaying intricate graphs, such as social networks, biological networks, and more. In order to obtain practical expertise using well-known graph analytics tools like igraph and sna in R, we will deal with real-world data sets.

This project's primary goal is to apply graph analytics to a sizable and intricate data set. In order to prepare and clean data for analysis, we will learn how to do a variety of graph algorithms, including clustering and centrality analysis, and visualise the outcomes. We will also look at some of the speed and scalability problems that come with working with huge graphs.

Configuration:
1. Launch RStudio (this requires that you have installed and downloaded R beforehand)
2. Use the Edit menu to clear the console
3. From the Session menu, clear the workspace
4. Set up igraph
5. Declare it as a Library

This is an introductory project to get used to using R.

1. Data Set: soc-Epinions1_adj.tsv (on Blackboard)

Here we will use the data set that provided in the Class Project Description
i.e., 'soc-Epinions_adj.tsv.

A graph of links of opinions from the SOC-E website. Names have been removed and replaced by numbers.

Format of each row is <node-1>, <node-2>, # Edges.

For this data #Edges is 1 for each row, so it can be removed, once you load the data set as a matrix or a data frame. There are approximately 10 million nodes.

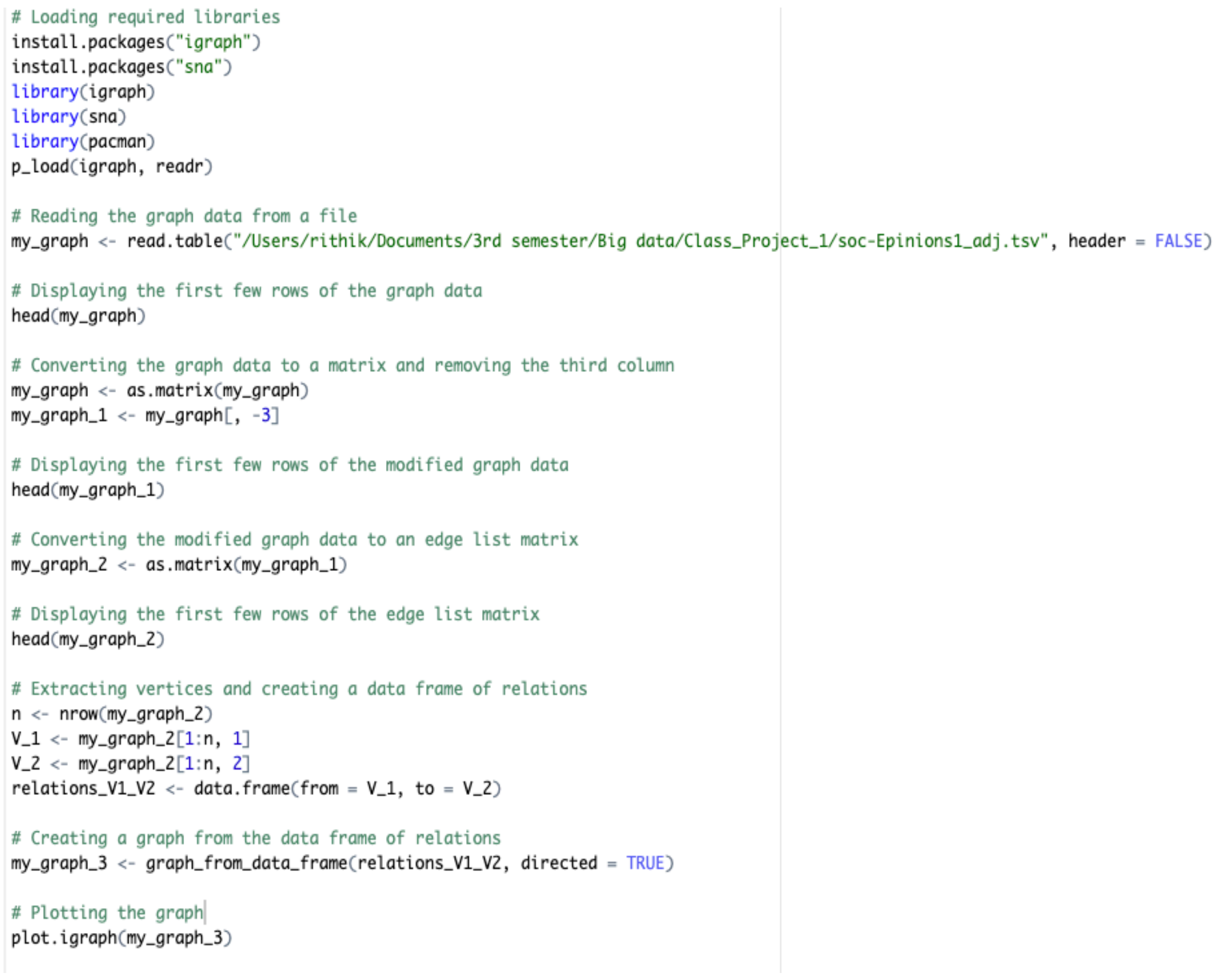
2. Install the igraph package from one of the CRAN mirrors.
a. Import the specified data set

One way to do this is the following procedure:

1. opinions<-read.table(<<data set>>)   assuming you have set the working directory
2. convert to matrix: optab <-as.matrix(opinions)
3. extract vectors: v1 <- optab[1:n,1], remaining vectors have similar form. This gets the vertices as separate vectors
4. relations<- data.frame(from=v1,to=v2)
5. g<-graph.data.frame(relations,directed=TRUE) need to have installed igraph
6. plot(g)

Our code to plot a graph without simplification is shown in below screenshot (Figure-2.1).

Figure: 2.1



```r
# Loading required libraries
install.packages("igraph")
install.packages("sna")
library(igraph)
library(sna)
library(pacman)
p_load(igraph, readr)

# Reading the graph data from a file
my_graph <- read.table("/Users/rithik/Documents/3rd semester/Big data/Class_Project_1/soc-Epinions1_adj.tsv", header = FALSE)

# Displaying the first few rows of the graph data
head(my_graph)

# Converting the graph data to a matrix and removing the third column
my_graph <- as.matrix(my_graph)
my_graph_1 <- my_graph[, -3]

# Displaying the first few rows of the modified graph data
head(my_graph_1)

# Converting the modified graph data to an edge list matrix
my_graph_2 <- as.matrix(my_graph_1)

# Displaying the first few rows of the edge list matrix
head(my_graph_2)

# Extracting vertices and creating a data frame of relations
n <- nrow(my_graph_2)
V_1 <- my_graph_2[1:n, 1]
V_2 <- my_graph_2[1:n, 2]
relations_V1_V2 <- data.frame(from = V_1, to = V_2)

# Creating a graph from the data frame of relations
my_graph_3 <- graph_from_data_frame(relations_V1_V2, directed = TRUE)

# Plotting the graph
plot.igraph(my_graph_3)
```

The code in Figure 2.1 works as The code reads a graph from a file in the form of an adjacency matrix. It then removes the third column from the matrix, converts the modified matrix into an edge list matrix, and creates a graph from the edge list matrix. Finally, it plots the graph. Here's a step-by-step breakdown of the code:

1. `my_graph <- read.table("/Users/rithik/Documents/3rd semester/Big data/Class_Project_1/soc-Epinions1_adj.tsv", header = FALSE)`: This line reads the graph data from a file named "soc-Epinions1_adj.tsv" located in the specified directory. The data is read as a table (matrix) with no header.

2. `my_graph <- as.matrix(my_graph)`: This line converts the graph data into a matrix.

3. `my_graph_1 <- my_graph[, -3]`: This line removes the third column from the matrix.

4. `my_graph_2 <- as.matrix(my_graph_1)`: This line converts the modified matrix into an edge list matrix.

5. `n <- nrow(my_graph_2)`: This line calculates the number of rows in the edge list matrix.

6. `V_1 <- my_graph_2[1:n, 1]`: This line extracts the first vertices from the edge list matrix.

7. `V_2 <- my_graph_2[1:n, 2]`: This line extracts the second vertices from the edge list matrix.

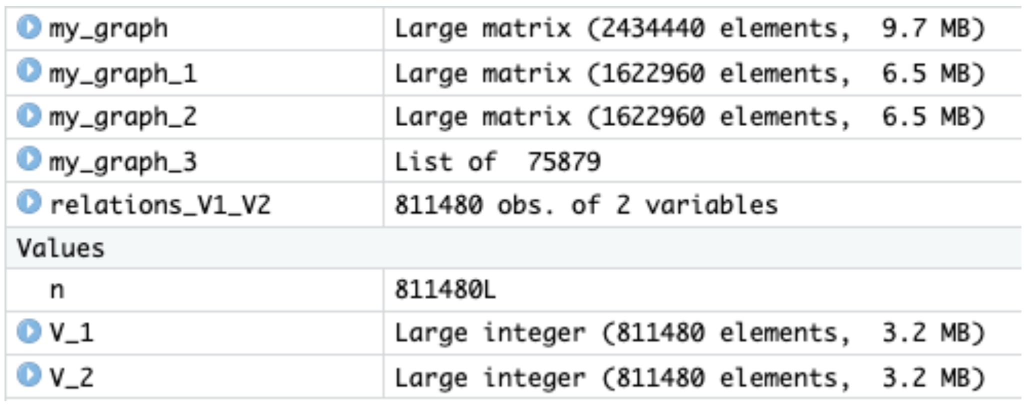
8. `relations_V1_V2 <- data.frame(from = V_1, to = V_2)`: This line creates a data frame of relations, where each row represents a relation between two vertices.

9. `my_graph_3 <- graph_from_data_frame(relations_V1_V2, directed = TRUE)`: This line creates a graph from the data frame of relations. The graph is directed, meaning that the direction of the relations is taken into account.

10. `plot.igraph(my_graph_3)`: This line plots the graph.

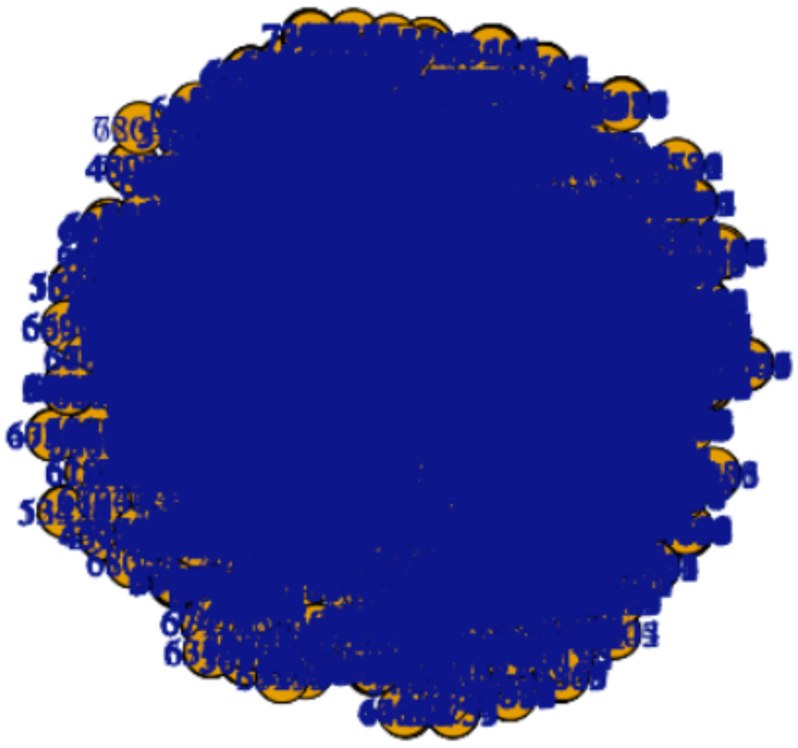
The resulting plot is a visual representation of the graph, where vertices are represented by circles and relations are represented by lines connecting the corresponding vertices.

Figure: 2.2: List of Variables and their Values.



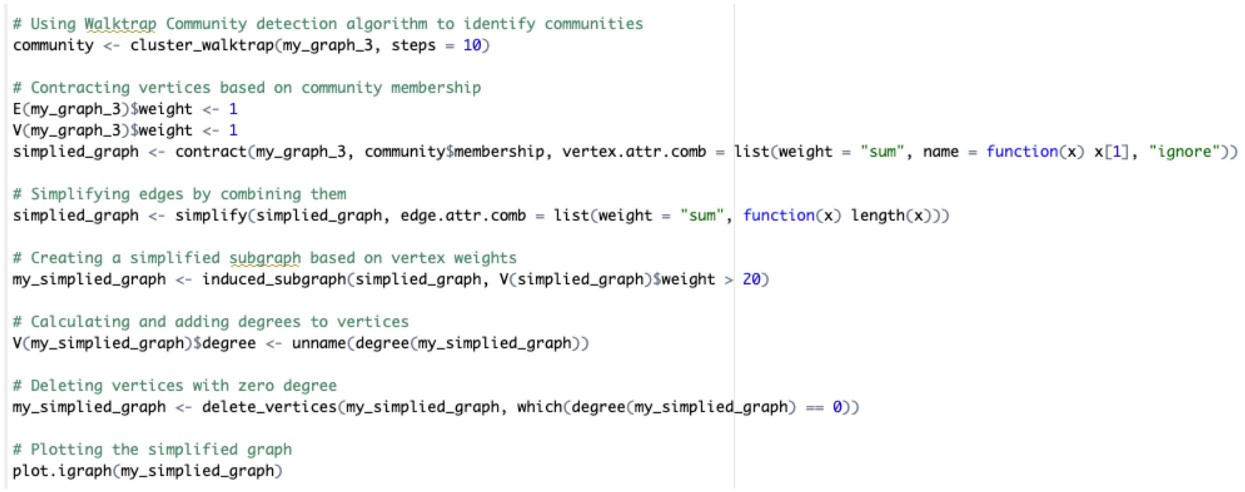
my_graph	Large matrix (2434440 elements, 9.7 MB)
my_graph_1	Large matrix (1622960 elements, 6.5 MB)
my_graph_2	Large matrix (1622960 elements, 6.5 MB)
my_graph_3	List of 75879
relations_V1_V2	811480 obs. of 2 variables
Values	
n	811480L
V_1	Large integer (811480 elements, 3.2 MB)
V_2	Large integer (811480 elements, 3.2 MB)

Figure: 2.3: The Graph



b. Determine how to create a graph and plot. Show the plot in your report.
We have simplified the graph to execute other functions easily.

Simplification of graph:



```
# Using Walktrap Community detection algorithm to identify communities
community <- cluster_walktrap(my_graph_3, steps = 10)

# Contracting vertices based on community membership
E(my_graph_3)$weight <- 1
V(my_graph_3)$weight <- 1
simplied_graph <- contract(my_graph_3, community$membership, vertex.attr.comb = list(weight = "sum", name = function(x) x[1], "ignore"))

# Simplifying edges by combining them
simplied_graph <- simplify(simplied_graph, edge.attr.comb = list(weight = "sum", function(x) length(x)))

# Creating a simplified subgraph based on vertex weights
my_simplied_graph <- induced_subgraph(simplied_graph, V(simplied_graph)$weight > 20)

# Calculating and adding degrees to vertices
V(my_simplied_graph)$degree <- unname(degree(my_simplied_graph))

# Deleting vertices with zero degree
my_simplied_graph <- delete_vertices(my_simplied_graph, which(degree(my_simplied_graph) == 0))

# Plotting the simplified graph
plot.igraph(my_simplied_graph)
```

This code is processing a graph and identifying communities within it, followed by simplification and visualization. Here's a breakdown of what each step does:

1. Community Detection:

- cluster_walktrap(my_graph_3, steps = 10): This line uses the Walktrap algorithm to identify communities within the graph represented by my_graph_3. The algorithm iteratively simulates "walkers" moving across the graph, favouring edges connecting vertices within the same community. After 10 steps, the algorithm assigns each vertex to a community based on where the walkers ended up.

2. Contracting Vertices:

- E(my_graph_3)$weight <- 1: This line sets the weight of all edges in the graph to 1.
- V(my_graph_3)$weight <- 1: This line sets the weight of all vertices in the graph to 1.
- contract(my_graph_3, community$membership, vertex.attr.comb = list(weight = "sum", name = function(x) x[1], "ignore")): This line simplifies the graph by merging vertices that belong to the same community. Vertices are merged by summing their weights (weight = "sum") and keeping the name of the first vertex encountered (name = function(x) x[1]). Any other vertex attributes are ignored ("ignore").

3. Simplifying Edges:

- simplify (simplified_graph, edge.attr.comb = list(weight = "sum", function(x) length(x))): This line further simplifies the graph by merging edges that connect the same pair of contracted vertices. The weight of the merged edge becomes the sum of the original edge weights (weight = "sum"), and the number of edges between the vertices is stored as a new attribute (function(x) length(x))

4. Subgraph Creation:

- induced_subgraph(simplified_graph, V(simplified_graph)$weight > 20): This line creates a new subgraph containing only vertices with a weight greater than 20. This likely focuses on vertices that belong to larger communities.

5. Calculating Degrees:

- V(my_simplified_graph)$degree <- unname(degree(my_simplified_graph)): This line calculates the degree (number of connections) for each vertex in the subgraph and stores it as an attribute named "degree".

6. Removing Isolated Vertices:

- delete_vertices(my_simplified_graph, which(degree(my_simplified_graph) == 0)): This line removes any vertices that have no connections (degree of 0) from the subgraph.

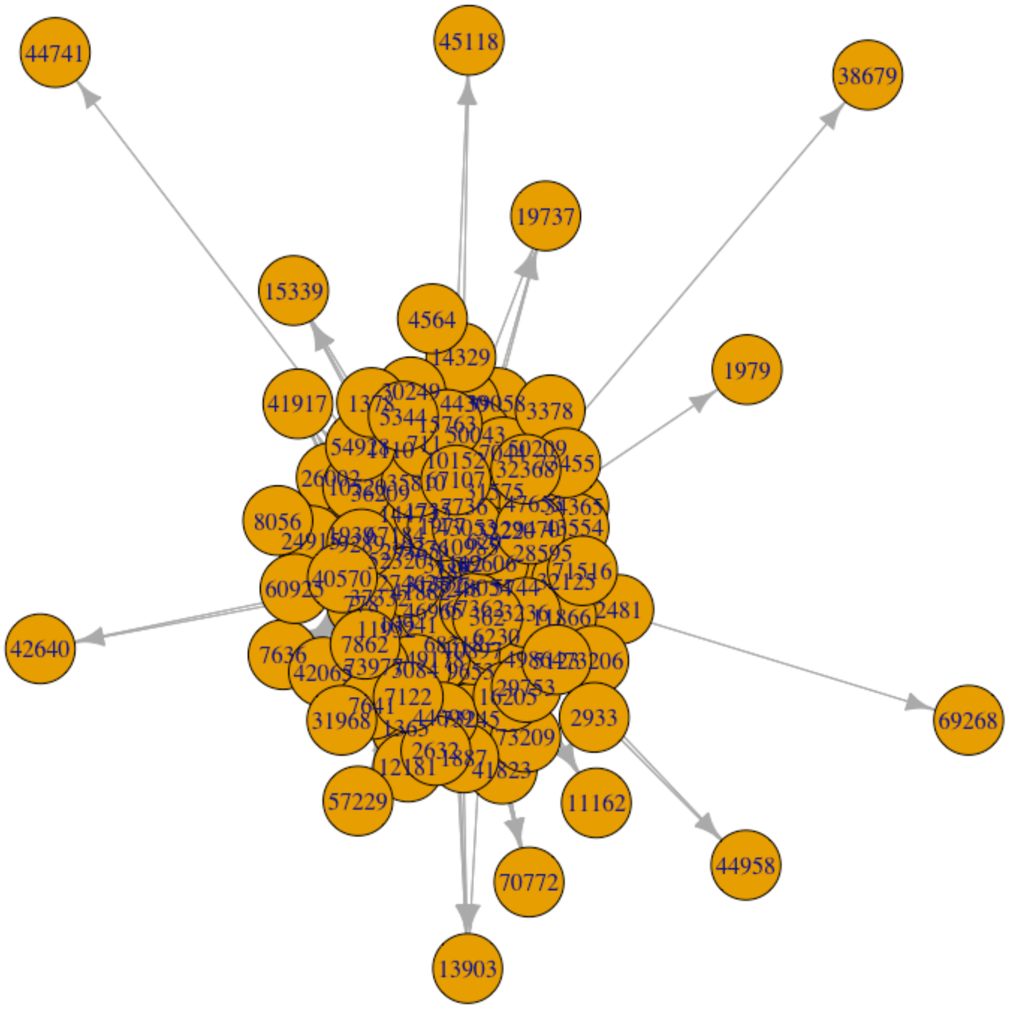
7. Visualization:

- plot.igraph(my_simplified_graph): This line finally plots the simplified graph, allowing you to visually inspect the identified communities and their connections.

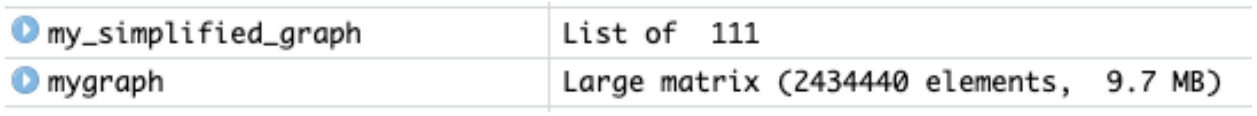
Overall, this code utilizes the Walktrap algorithm to identify communities in a graph, then simplifies the graph by merging vertices within communities and removing less connected elements. Finally, it visualizes the resulting simplified graph to help understand the community structure.

Finally, the simplified graph my_simplified_graph is plotted using the plot.igraph() function.

Plot:



Variables:

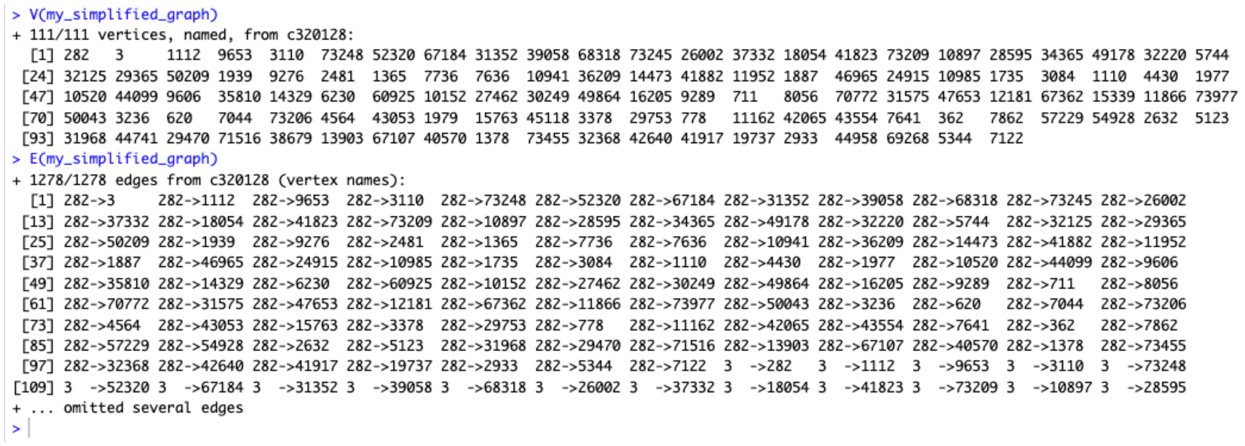


my_simplified_graph	List of 111
mygraph	Large matrix (2434440 elements, 9.7 MB)

3. Apply the functions that I have shown in the *Introduction to Graph Analytics* document on Blackboard on the graph generated from the data set.

a. Present the results in your write-up as screen shots.
b. Explain what the function could tell you about the problem domain.

Displaying Edges and Vertices:-



```
> V(my_simplified_graph)
+ 111/111 vertices, named, from c320128:
  [1] 282   3     1112  9653  3110  73248 52320 67184 31352 39058 68318 73245 26002 37332 18054 41823 73209 10897 28595 34365 49178 32220 5744
 [24] 32125 29365 50209 1939  9276  2481  1365  7736  7636  10941 36209 14473 41882 11952 1887  46965 24915 10985 1735  3084  1110  4430  1977
 [47] 10520 44099 9606  35810 14329 6230  60925 10152 27462 30249 49864 16205 9289  711   8056  70772 31575 47653 12181 67362 15339 11866 73977
 [70] 50043 3236  620   7044  73206 4564  43053 1979  15763 45118 3378  29753 778   11162 42065 43554 7641  362   7862  57229 54928 2632  5123
 [93] 31968 44741 29470 71516 38679 13903 67107 40570 1378  73455 32368 42640 41917 19737 2933  44958 69268 5344  7122
> E(my_simplified_graph)
+ 1278/1278 edges from c320128 (vertex names):
  [1] 282->3     282->1112  282->9653  282->3110  282->73248 282->52320 282->67184 282->31352 282->39058 282->68318 282->73245 282->26002
 [13] 282->37332 282->18054 282->41823 282->73209 282->10897 282->28595 282->34365 282->49178 282->32220 282->5744  282->32125 282->29365
 [25] 282->50209 282->1939  282->9276  282->2481  282->1365  282->7736  282->7636  282->10941 282->36209 282->14473 282->41882 282->11952
 [37] 282->1887  282->46965 282->24915 282->10985 282->1735  282->3084  282->1110  282->4430  282->1977  282->10520 282->44099 282->9606
 [49] 282->35810 282->14329 282->6230  282->60925 282->10152 282->27462 282->30249 282->49864 282->16205 282->9289  282->711   282->8056
 [61] 282->70772 282->31575 282->47653 282->12181 282->67362 282->11866 282->73977 282->50043 282->3236  282->620   282->7044  282->73206
 [73] 282->4564  282->43053 282->15763 282->3378  282->29753 282->778   282->11162 282->42065 282->43554 282->7641  282->362   282->7862
 [85] 282->57229 282->54928 282->2632  282->5123  282->31968 282->29470 282->71516 282->13903 282->67107 282->40570 282->1378  282->73455
 [97] 282->32368 282->42640 282->41917 282->19737 282->2933  282->5344  282->7122  3   ->282   3   ->1112  3   ->9653  3   ->3110  3   ->73248
[109] 3   ->52320 3   ->67184 3   ->31352 3   ->39058 3   ->68318 3   ->26002 3   ->37332 3   ->18054 3   ->41823 3   ->73209 3   ->10897 3   ->28595
+ ... omitted several edges
> 
```

Figure 3.1

In Figure 3.1,

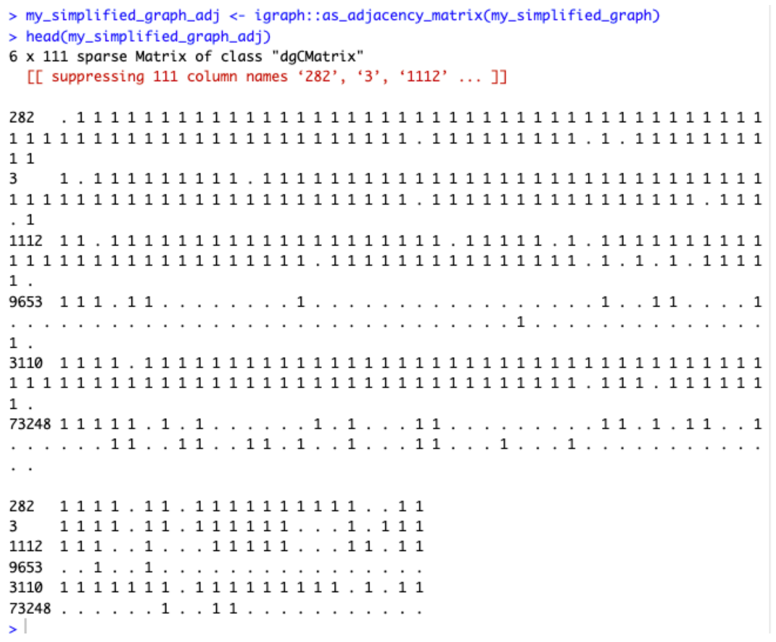
- V(my_simplified_graph) #displays vertices of graph

This function will give the vertices of the graph by taking an graph object as an input.

- E(my_simplified_graph)#displays edges of graph

This function will give the vertices of the graph by taking an graph object as an input.

Adjacency Matrix:-



```
> my_simplified_graph_adj <- igraph::as_adjacency_matrix(my_simplified_graph)
> head(my_simplified_graph_adj)
6 x 111 sparse Matrix of class "dgCMatrix"
  [[ suppressing 111 column names ‘282’, ‘3’, ‘1112’ ... ]]

282   . 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 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 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 1
3     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 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 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
1112  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 . 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 . 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . 1 . 1 . 1 . 1 1 1 1
1 .
9653  1 1 1 . 1 1 . . . . . . . . 1 . . . . . . . . . . . . . . . . . 1 . . 1 1 . . . . 1
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1 . . . . . . . . . . . . . .
1 .
3110  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 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 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 .
73248 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 . . . 1 . . . . . . . . . . .
. .

282   1 1 1 1 . 1 1 . 1 1 1 1 1 1 1 1 1 1 . . 1 1
3     1 1 1 1 . 1 1 . 1 1 1 1 1 1 . . . 1 . 1 1 1
1112  1 1 1 . . 1 . . . 1 1 1 1 1 . . . 1 1 . 1 1
9653  . . 1 . . 1 . . . . . . . . . . . . . . . .
3110  1 1 1 1 1 1 1 . 1 1 1 1 1 1 1 1 1 . 1 . 1 1
73248 . . . . . . 1 . . 1 1 . . . . . . . . . . .
> 
```

Figure 3.2

In Figure 3.2,

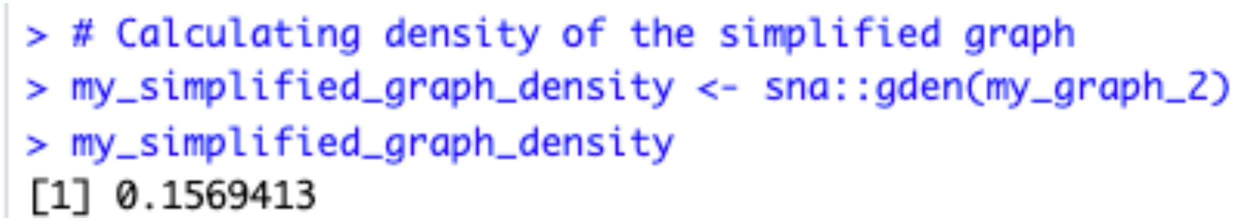
- my_simplified_graph.adj=igraph::get.adjacency(my_simplified_graph)

This function helps to find the adjacency matrix of graph by taking an graph object as an input.

- head(my_simplified_graph.adj)

This function helps to return the first few rows of the dataset.

Density:-



```
> # Calculating density of the simplified graph
> my_simplified_graph_density <- sna::gden(my_graph_2)
> my_simplified_graph_density
[1] 0.1569413
```

Figure 3.3

In Figure 3.3,

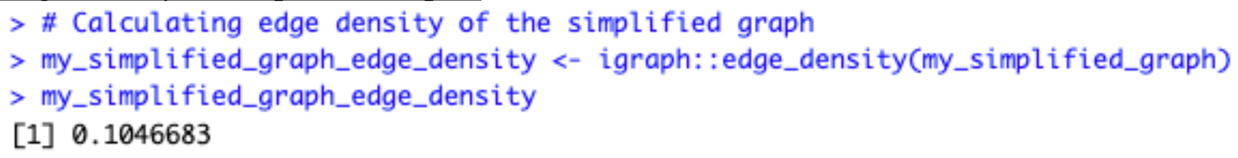
- my_simplified_graph.density=sna::gden(mygraph2) #finding density of the graph

A graph object is entered into the “gden” function, which then determines the graph’s density. The resultant value is a number in the range of 0 to 1, where 0 denotes the absence of edges in the graph and 1 denotes full connectivity (i.e., the presence of all conceivable edges).

- my_simplified_graph.density

This function calculates the density of edges in the graph.

Edge Density of Simplified Graph:-



```
> # Calculating edge density of the simplified graph
> my_simplified_graph_edge_density <- igraph::edge_density(my_simplified_graph)
> my_simplified_graph_edge_density
[1] 0.1046683
```

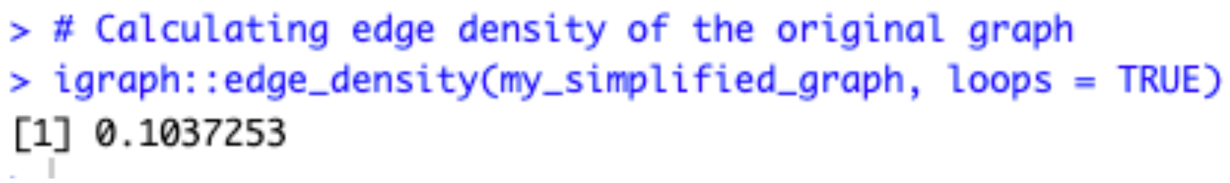
Figure 3.4

In Figure 3.4,

- igraph::edge_density(my_simplified_graph)

The edge_density() function takes a graph object as its input and returns a floating-point value representing the edge density of the graph.

Edge Density of Original Graph:-



```
> # Calculating edge density of the original graph
> igraph::edge_density(my_simplified_graph, loops = TRUE)
[1] 0.1037253
```

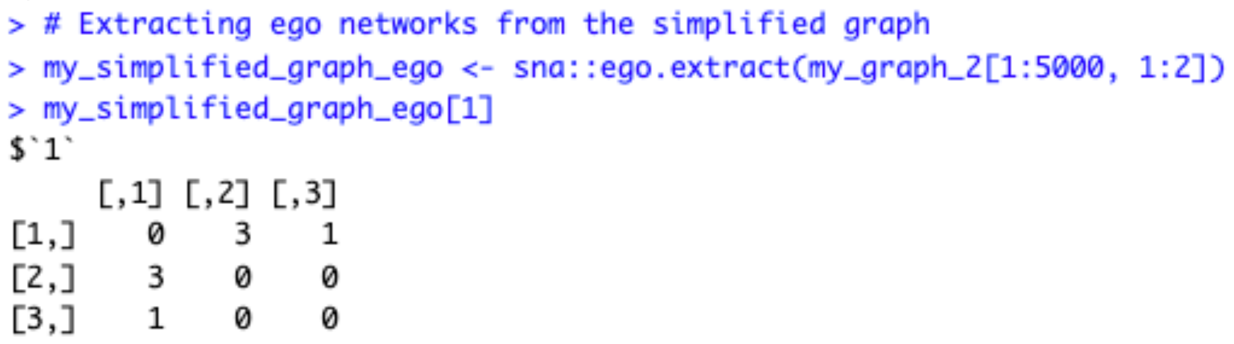
Figure 3.5

In Figure 3.5,

- igraph::edge_density(my_simplified_graph, loops = TRUE)

This function will compute the edge density of the graph, which is the ratio of the number of edges to the maximum possible number of edges, including loops (self-connections).

Ego Networks:-



```
> # Extracting ego networks from the simplified graph
> my_simplified_graph_ego <- sna::ego.extract(my_graph_2[1:5000, 1:2])
> my_simplified_graph_ego[1]
$`1`
     [,1] [,2] [,3]
[1,]    0    3    1
[2,]    3    0    0
[3,]    1    0    0
```

Figure 3.6

In Figure 3.6,

- my_simplified_graph ego=sna::ego.extract(mygraph2[1:5000,1:2])

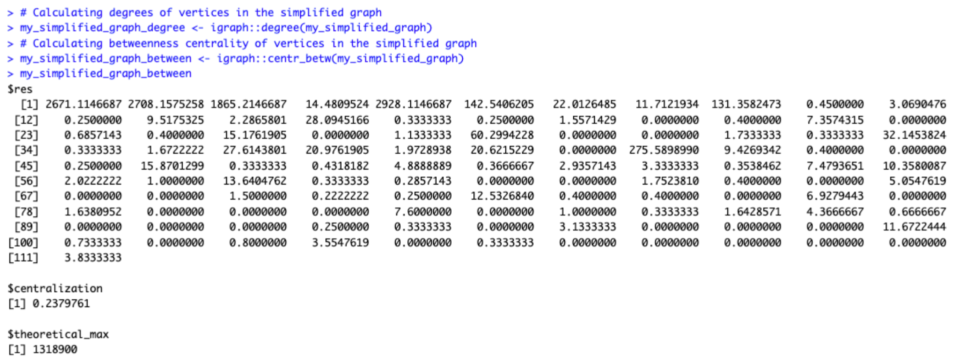
This function extracts egos (nodes) from **mygraph2** within the range of 1 to 5000 for columns 1 and 2, creating an ego network in **my_simplified_graph**.

- my_simplified_graph.ego[1]

This function accesses the first ego network within **my_simplified_graph**.

The ego() function takes a graph object and a vertex ID as its inputs, and returns a subgraph object representing the ego network of the specified vertex.

Degrees of Vertices:-



```
> # Calculating degrees of vertices in the simplified graph
> my_simplified_graph_degree <- igraph::degree(my_simplified_graph)
> # Calculating betweenness centrality of vertices in the simplified graph
> my_simplified_graph_between <- igraph::centr_betw(my_simplified_graph)
> my_simplified_graph_between
$res
  [1] 2671.1146687 2708.1575258 1865.2146687   14.4809524 2928.1146687  142.5406205   22.0126485   11.7121934  131.3582473    0.4500000    3.0690476
 [12]    0.2500000    9.5175325    2.2865801   28.0945166    0.3333333    0.2500000    1.5571429    0.0000000    0.4000000    7.3574315    0.0000000
 [23]    0.6857143    0.4000000   15.1761905    0.0000000    1.1333333   60.2994228    0.0000000    0.0000000    1.7333333    0.3333333   32.1453824
 [34]    0.3333333    1.6722222   27.6143801   20.9761905    1.9728938   20.6215229    0.0000000  275.5898990    9.4269342    0.4000000    0.0000000
 [45]    0.2500000   15.8701299    0.3333333    0.4318182    4.8888889    0.3666667    2.9357143    3.3333333    0.3538462    7.4793651   10.3580087
 [56]    2.0222222    1.0000000   13.6404762    0.3333333    0.2857143    0.0000000    0.0000000    1.7523810    0.4000000    0.0000000    5.0547619
 [67]    0.0000000    0.0000000    1.5000000    0.2222222    0.2500000   12.5326840    0.4000000    0.4000000    0.0000000    6.9279443    0.0000000
 [78]    1.6380952    0.0000000    0.0000000    0.0000000    7.6000000    0.0000000    1.0000000    0.3333333    1.6428571    4.3666667    0.6666667
 [89]    0.0000000    0.0000000    0.0000000    0.2500000    0.3333333    0.0000000    3.1333333    0.0000000    0.0000000    0.0000000   11.6722444
[100]    0.7333333    0.0000000    0.8000000    3.5547619    0.0000000    0.3333333    0.0000000    0.0000000    0.0000000    0.0000000    0.0000000
[111]    3.8333333

$centralization
[1] 0.2379761

$theoretical_max
[1] 1318900
```

Figure 3.7

In Figure 3.7,

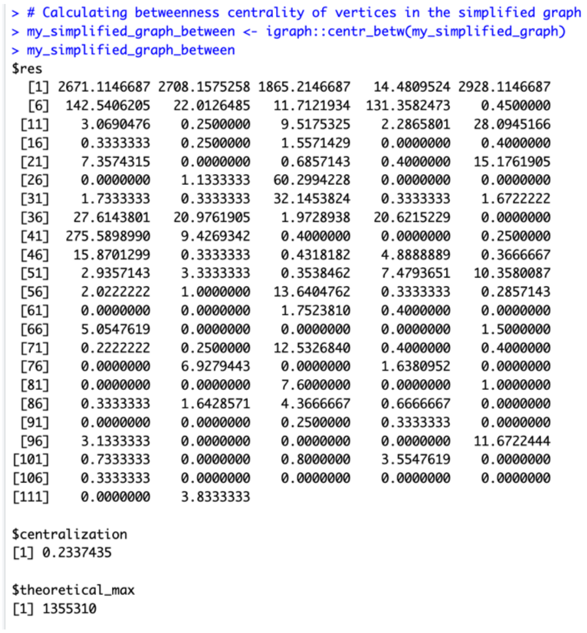
- my_simplified_graph_degree=igraph::degree(my_simplified_graph)

The degree() function calculates the degree (number of connections) of each node in the graph. This assigns the degrees to the variable **my_simplified_graph_degree**, providing a vector where each element corresponds to the degree of a node in **my_simplified_graph**.

- my_simplified_graph_between=igraph::centr_betw(my_simplified_graph)

The centr_betw() function computes the betweenness centrality for each node in the graph, indicating how often a node lies on the shortest path between other nodes. This assigns the betweenness centralities to the variable **my_simplified_graph_between**.

Betweenness Centrality:-



```
> # Calculating betweenness centrality of vertices in the simplified graph
> my_simplified_graph_between <- igraph::centr_betw(my_simplified_graph)
> my_simplified_graph_between
$res
  [1] 2671.1146687 2708.1575258 1865.2146687   14.4809524 2928.1146687
  [6]  142.5406205   22.0126485   11.7121934  131.3582473    0.4500000
 [11]    3.0690476    0.2500000    9.5175325    2.2865801   28.0945166
 [16]    0.3333333    0.2500000    1.5571429    0.0000000    0.4000000
 [21]    7.3574315    0.0000000    0.6857143    0.4000000   15.1761905
 [26]    0.0000000    1.1333333   60.2994228    0.0000000    0.0000000
 [31]    1.7333333    0.3333333   32.1453824    0.3333333    1.6722222
 [36]   27.6143801   20.9761905    1.9728938   20.6215229    0.0000000
 [41]  275.5898990    9.4269342    0.4000000    0.0000000    0.2500000
 [46]   15.8701299    0.3333333    0.4318182    4.8888889    0.3666667
 [51]    2.9357143    3.3333333    0.3538462    7.4793651   10.3580087
 [56]    2.0222222    1.0000000   13.6404762    0.3333333    0.2857143
 [61]    0.0000000    0.0000000    1.7523810    0.4000000    0.0000000
 [66]    5.0547619    0.0000000    0.0000000    0.0000000    1.5000000
 [71]    0.2222222    0.2500000   12.5326840    0.4000000    0.4000000
 [76]    0.0000000    6.9279443    0.0000000    1.6380952    0.0000000
 [81]    0.0000000    0.0000000    7.6000000    0.0000000    1.0000000
 [86]    0.3333333    1.6428571    4.3666667    0.6666667    0.0000000
 [91]    0.0000000    0.0000000    0.2500000    0.3333333    0.0000000
 [96]    3.1333333    0.0000000    0.0000000    0.0000000   11.6722444
[101]    0.7333333    0.0000000    0.8000000    3.5547619    0.0000000
[106]    0.3333333    0.0000000    0.0000000    0.0000000    0.0000000
[111]    0.0000000    3.8333333

$centralization
[1] 0.2337435

$theoretical_max
[1] 1355310
```

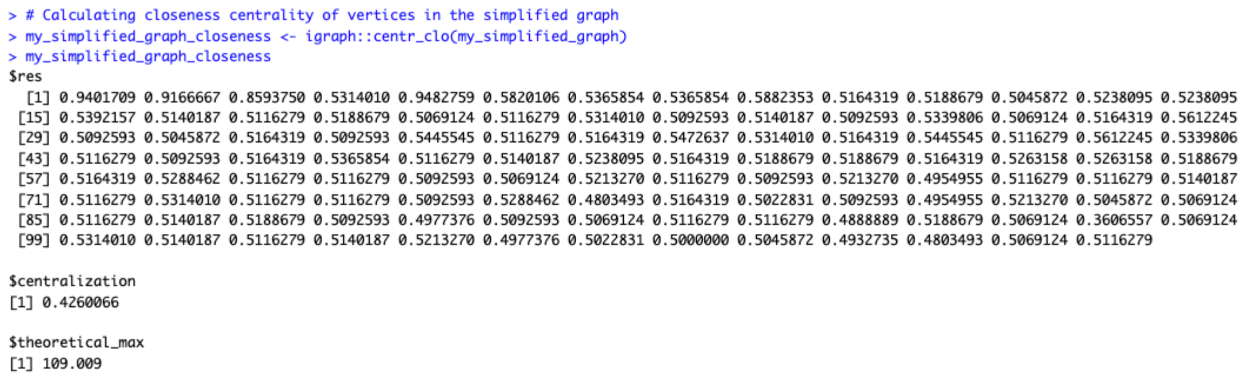
Figure 3.8

In Figure 3.8,

- my_simplified_graph_between <- igraph::centr_betw(my_simplified_graph)
- my_simplified_graph_between

The betweenness centrality of every vertex in the simplified graph my_simplified_graph is determined by this code. By counting the number of shortest paths that pass through a vertex, betweenness centrality measures the vertex's importance and suggests how much of an impact it may have on communication or information flow within the graph. The betweenness centrality scores for each vertex are included in the resulting my_simplified_graph_between object, which tells how important each vertex is in terms of influence or communication within the network.

<u>Closeness Centrality:-</u>



```
> # Calculating closeness centrality of vertices in the simplified graph
> my_simplified_graph_closeness <- igraph::centr_clo(my_simplified_graph)
> my_simplified_graph_closeness
$res
  [1] 0.9401709 0.9166667 0.8593750 0.5314010 0.9482759 0.5820106 0.5365854 0.5365854 0.5882353 0.5164319 0.5188679 0.5045872 0.5238095 0.5238095
 [15] 0.5392157 0.5140187 0.5116279 0.5188679 0.5069124 0.5116279 0.5314010 0.5092593 0.5140187 0.5092593 0.5339806 0.5069124 0.5164319 0.5612245
 [29] 0.5092593 0.5045872 0.5164319 0.5092593 0.5445545 0.5116279 0.5164319 0.5472637 0.5314010 0.5164319 0.5445545 0.5116279 0.5612245 0.5339806
 [43] 0.5116279 0.5092593 0.5164319 0.5365854 0.5116279 0.5140187 0.5238095 0.5164319 0.5188679 0.5188679 0.5164319 0.5263158 0.5263158 0.5188679
 [57] 0.5164319 0.5288462 0.5116279 0.5116279 0.5092593 0.5069124 0.5213270 0.5116279 0.5092593 0.5213270 0.4954955 0.5116279 0.5116279 0.5140187
 [71] 0.5116279 0.5314010 0.5116279 0.5116279 0.5092593 0.5288462 0.4803493 0.5164319 0.5022831 0.5092593 0.4954955 0.5213270 0.5045872 0.5069124
 [85] 0.5116279 0.5140187 0.5188679 0.5092593 0.4977376 0.5092593 0.5069124 0.5116279 0.5116279 0.4888889 0.5188679 0.5069124 0.3606557 0.5069124
 [99] 0.5314010 0.5140187 0.5116279 0.5140187 0.5213270 0.4977376 0.5022831 0.5000000 0.5045872 0.4932735 0.4803493 0.5069124 0.5116279

$centralization
[1] 0.4260066

$theoretical_max
[1] 109.009
```

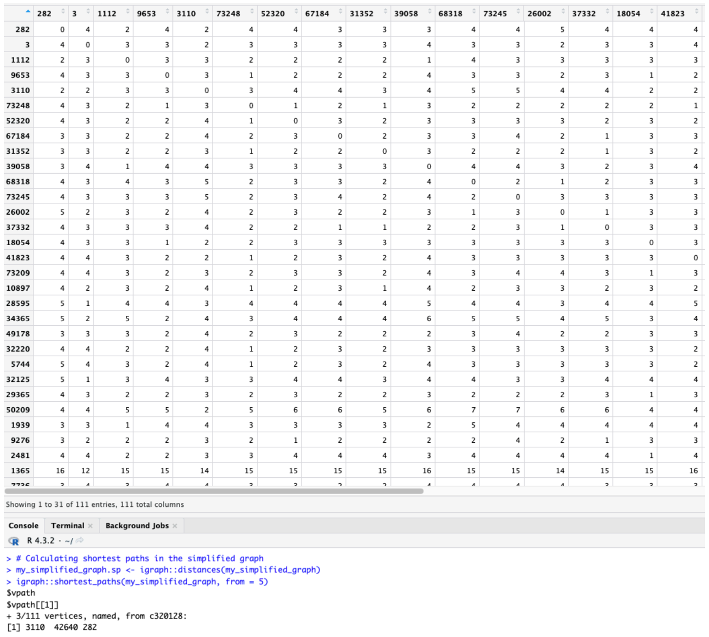
Figure 3.9

In Figure 3.9,

- my_simplified_graph_closeness=igraph::centr_clo(my_simplified_graph)
- my_simplified_graph_closeness

The function `my_simplified_graph_closeness=igraph::centr_clo(my_simplified_graph)` calculates the closeness centrality for each node in the graph, representing how close a node is to all other nodes in the network. The resulting `my_simplified_graph_closeness` holds these centrality scores for each node. Higher scores indicate nodes that are closer to all other nodes, potentially having greater influence or access to information within the network.

Shortest Paths:-



	282	3	1112	9653	3110	73248	52320	67184	31352	39058	68318	73245	26002	37332	18054	41823
282	0	4	2	4	2	4	4	3	3	3	4	4	5	4	4	4
3	4	0	3	3	2	3	3	3	3	4	3	3	2	3	3	4
1112	2	3	0	3	3	2	2	2	2	1	4	3	3	3	3	3
9653	4	3	3	0	3	1	2	2	2	4	3	3	2	3	1	2
3110	2	2	3	3	0	3	4	4	3	4	5	5	4	4	2	2
73248	4	3	2	1	3	0	1	2	1	3	2	2	2	2	2	1
52320	4	3	2	2	4	1	0	3	2	3	3	3	3	2	3	2
67184	3	3	2	2	4	2	3	0	2	3	3	4	2	1	3	3
31352	3	3	2	2	3	1	2	2	0	3	2	2	2	1	3	2
39058	3	4	1	4	4	3	3	3	3	0	4	4	3	2	3	4
68318	4	3	4	3	5	2	3	3	2	4	0	2	1	2	3	3
73245	4	3	3	3	5	2	3	4	2	4	2	0	3	3	3	3
26002	5	2	3	2	4	2	3	2	2	3	1	3	0	1	3	3
37332	4	3	3	3	4	2	2	1	1	2	2	3	1	0	3	3
18054	4	3	3	1	2	2	3	3	3	3	3	3	3	3	0	3
41823	4	4	3	2	2	1	2	3	2	4	3	3	3	3	3	0
73209	4	4	3	2	3	2	3	3	2	4	3	4	4	3	1	3
10897	4	2	3	2	4	1	2	3	1	4	2	3	3	2	3	2
28595	5	1	4	4	3	4	4	4	4	5	4	4	3	4	4	5
34365	5	2	5	2	4	3	4	4	4	6	5	5	4	5	3	4
49178	3	3	3	2	4	2	3	2	2	2	3	4	2	2	3	3
32220	4	4	2	2	4	1	2	3	2	3	3	3	3	3	3	2
5744	5	4	3	2	4	1	2	3	2	4	3	3	3	3	3	2
32125	5	1	3	4	3	3	4	4	3	4	4	3	3	4	4	4
29365	4	3	2	2	3	2	3	2	2	3	2	2	2	3	1	3
50209	4	4	5	5	2	5	6	6	5	6	7	7	6	6	4	4
1939	3	3	1	4	4	3	3	3	3	2	5	4	4	4	4	4
9276	3	2	2	2	3	2	1	2	2	2	2	4	2	1	3	3
2481	4	4	2	2	3	3	4	4	4	3	4	4	4	4	1	4
1365	16	12	15	15	14	15	15	15	15	16	15	15	14	15	15	16

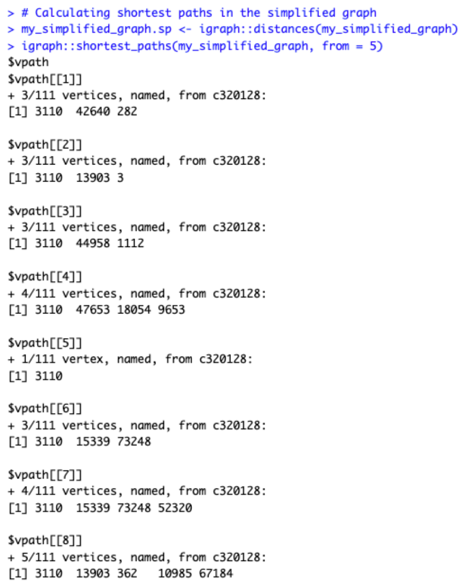
Showing 1 to 31 of 111 entries, 111 total columns

Console Terminal Background Jobs

R 4.3.2 · ~/

```
> # Calculating shortest paths in the simplified graph
> my_simplified_graph.sp <- igraph::distances(my_simplified_graph)
> igraph::shortest_paths(my_simplified_graph, from = 5)
$vpath
$vpath[[1]]
+ 3/111 vertices, named, from c320128:
[1] 3110  42640 282
```

Figure 3.10



```
> # Calculating shortest paths in the simplified graph
> my_simplified_graph.sp <- igraph::distances(my_simplified_graph)
> igraph::shortest_paths(my_simplified_graph, from = 5)
$vpath
$vpath[[1]]
+ 3/111 vertices, named, from c320128:
[1] 3110  42640 282

$vpath[[2]]
+ 3/111 vertices, named, from c320128:
[1] 3110  13903 3

$vpath[[3]]
+ 3/111 vertices, named, from c320128:
[1] 3110  44958 1112

$vpath[[4]]
+ 4/111 vertices, named, from c320128:
[1] 3110  47653 18054 9653

$vpath[[5]]
+ 1/111 vertex, named, from c320128:
[1] 3110

$vpath[[6]]
+ 3/111 vertices, named, from c320128:
[1] 3110  15339 73248

$vpath[[7]]
+ 4/111 vertices, named, from c320128:
[1] 3110  15339 73248 52320

$vpath[[8]]
+ 5/111 vertices, named, from c320128:
[1] 3110  13903 362   10985 67184
```

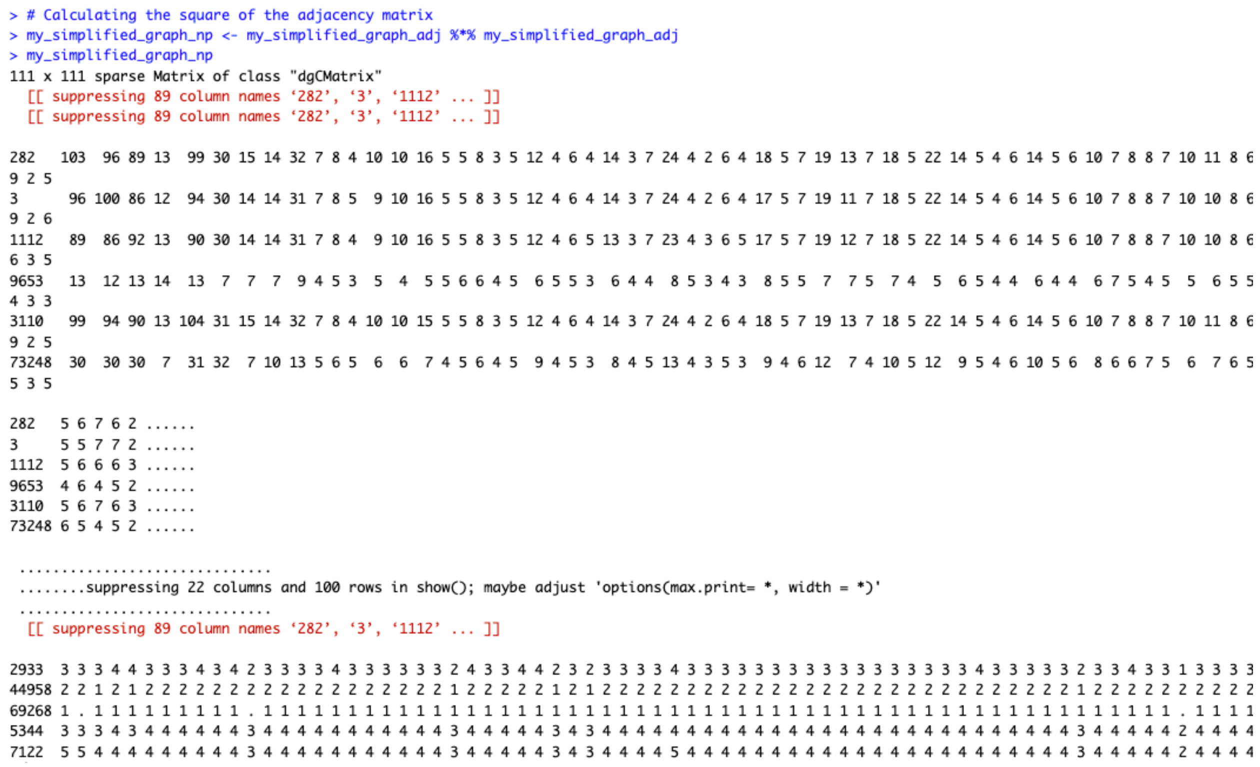
Figure 3.11

In Figure 3.10 & Figure 3.11,

- my_simplified_graph.sp=igraph::distances(my_simplified_graph)
- igraph::shortest_paths(my_simplified_graph, from=5)

The code calculates shortest paths in the simplified graph. **my_simplified_graph.sp <- igraph::distances(my_simplified_graph)** computes the shortest path lengths between all pairs of nodes in the graph, storing the distances in **my_simplified_graph.sp**. **igraph::shortest_paths(my_simplified_graph, from = 5)** then retrieves shortest paths starting from node 5 in the graph, providing information on the shortest paths from node 5 to all other nodes.

Square of the Adjacency Matrix:-



```
> # Calculating the square of the adjacency matrix
> my_simplified_graph_np <- my_simplified_graph_adj %*% my_simplified_graph_adj
> my_simplified_graph_np
111 x 111 sparse Matrix of class "dgCMatrix"
  [[ suppressing 89 column names ‘282’, ‘3’, ‘1112’ ... ]]
  [[ suppressing 89 column names ‘282’, ‘3’, ‘1112’ ... ]]

282   103  96 89 13  99 30 15 14 32 7 8 4 10 10 16 5 5 8 3 5 12 4 6 4 14 3 7 24 4 2 6 4 18 5 7 19 13 7 18 5 22 14 5 4 6 14 5 6 10 7 8 8 7 10 11 8 6
9 2 5
3      96 100 86 12  94 30 14 14 31 7 8 5  9 10 16 5 5 8 3 5 12 4 6 4 14 3 7 24 4 2 6 4 17 5 7 19 11 7 18 5 22 14 5 4 6 14 5 6 10 7 8 8 7 10 10 8 6
9 2 6
1112   89  86 92 13  90 30 14 14 31 7 8 4  9 10 16 5 5 8 3 5 12 4 6 5 13 3 7 23 4 3 6 5 17 5 7 19 12 7 18 5 22 14 5 4 6 14 5 6 10 7 8 8 7 10 10 8 6
6 3 5
9653   13  12 13 14  13  7  7  7  9 4 5 3  5  4  5 5 6 6 4 5  6 5 5 3  6 4 4  8 5 3 4 3  8 5 5  7  7 5  7 4  5  6 5 4 4  6 4 4  6 7 5 4 5  5  6 5 5
4 3 3
3110   99  94 90 13 104 31 15 14 32 7 8 4 10 10 15 5 5 8 3 5 12 4 6 4 14 3 7 24 4 2 6 4 18 5 7 19 13 7 18 5 22 14 5 4 6 14 5 6 10 7 8 8 7 10 11 8 6
9 2 5
73248  30  30 30  7  31 32  7 10 13 5 6 5  6  6  7 4 5 6 4 5  9 4 5 3  8 4 5 13 4 3 5 3  9 4 6 12  7 4 10 5 12  9 5 4 6 10 5 6  8 6 6 7 5  6  7 6 5
5 3 5

282   5 6 7 6 2 ......
3     5 5 7 7 2 ......
1112  5 6 6 6 3 ......
9653  4 6 4 5 2 ......
3110  5 6 7 6 3 ......
73248 6 5 4 5 2 ......

 ..............................
 ........suppressing 22 columns and 100 rows in show(); maybe adjust 'options(max.print= *, width = *)'
 ..............................
  [[ suppressing 89 column names ‘282’, ‘3’, ‘1112’ ... ]]

2933  3 3 3 4 4 3 3 3 4 3 4 2 3 3 3 3 4 3 3 3 3 3 3 2 4 3 3 4 4 2 3 2 3 3 3 3 4 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 4 3 3 3 3 3 2 3 3 4 3 3 1 3 3 3 3
44958 2 2 1 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 1 2 1 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 1 2 2 2 2 2 2 2 2 2 2
69268 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 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 1 1 1 1 1 1 1 1 . 1 1 1 1
5344  3 3 3 4 3 4 4 4 4 4 4 3 4 4 4 4 4 4 4 4 4 4 4 3 4 4 4 4 4 3 4 3 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 3 4 4 4 4 4 2 4 4 4 4
7122  5 5 4 4 4 4 4 4 4 4 4 3 4 4 4 4 4 4 4 4 4 4 4 3 4 4 4 4 4 3 4 3 4 4 4 4 5 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 3 4 4 4 4 4 2 4 4 4 4
```

Figure 3.12

In Figure 3.12,

- my_simplified_graph.np=my_simplified_graph.adj%*%my_simplified_graph.adj
- my_simplified_graph.np

The code computes the square of the adjacency matrix of the simplified graph. `my_simplified_graph_adj` represents the adjacency matrix of the graph. By performing matrix multiplication `%*%`, `my_simplified_graph_np` contains the result of multiplying the adjacency matrix by itself, yielding a new matrix where each element represents the number of paths of length 2 between corresponding nodes in the graph.

Histogram:-

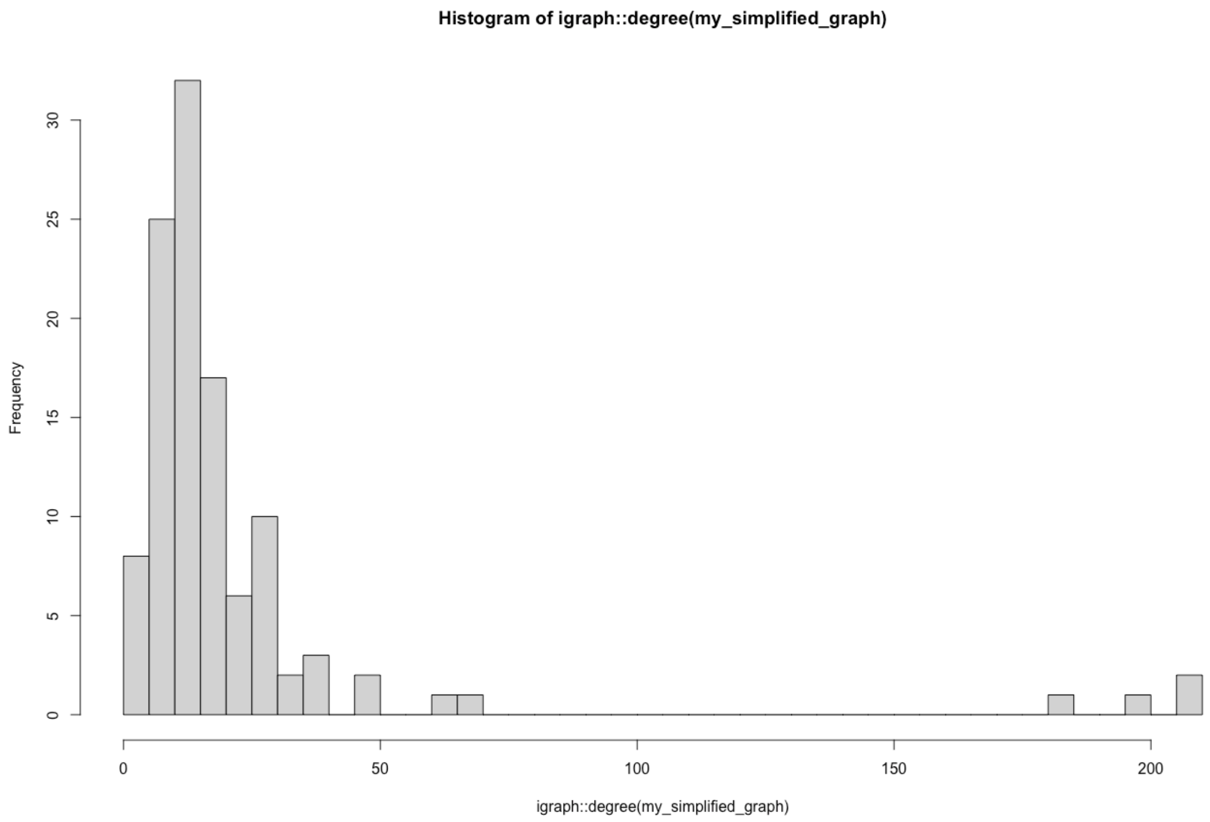


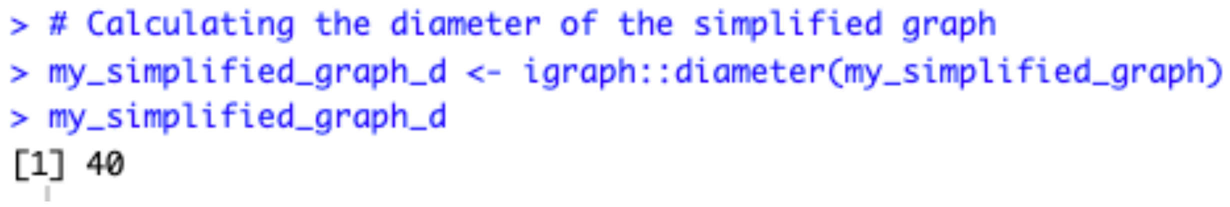
Figure 3.13

In Figure 3.13,

- hist(igraph::degree(my_simplified_graph),breaks = 40)

This function will plot the histogram of the degree of nodes. The code generates a histogram of node degrees in the simplified graph. It utilizes the `igraph::degree` function to calculate node degrees, then creates a histogram with 40 bins to visualize the distribution of node degrees within the graph.

Diameter of the Simplified Graph:-



```
> # Calculating the diameter of the simplified graph
> my_simplified_graph_d <- igraph::diameter(my_simplified_graph)
> my_simplified_graph_d
[1] 40
```

Figure 3.14

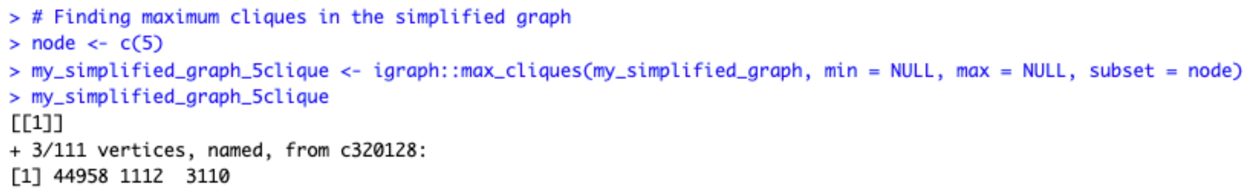
In Figure 3.14,

- my_simplified_graph.d=igraph::diameter(my_simplified_graph)
- my_simplified_graph_d

The code calculates the diameter of the simplified graph.
`my_simplified_graph.d=igraph::diameter(my_simplified_graph)` computes the longest shortest

path length between any pair of nodes in the graph, storing it in `my_simplified_graph_d`. This metric represents the maximum distance between any two nodes in the graph, indicating the overall size or extent of the network.

Clique of the Graph:-



```
> # Finding maximum cliques in the simplified graph
> node <- c(5)
> my_simplified_graph_5clique <- igraph::max_cliques(my_simplified_graph, min = NULL, max = NULL, subset = node)
> my_simplified_graph_5clique
[[1]]
+ 3/111 vertices, named, from c320128:
[1] 44958 1112  3110
```

Figure 3.15

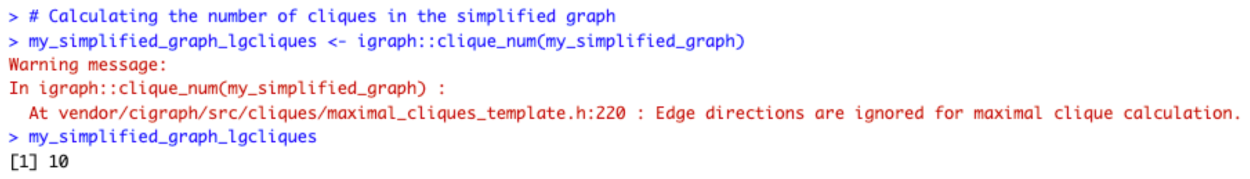
In Figure 3.15,

- node<-c(5)
- my_simplified_graph.5clique=igraph::max_cliques(my_simplified_graph,min=NULL,max=NULL,subset = node)
- my_simplified_graph.5clique

The code focuses on identifying cliques in the simplified graph containing node 5. It sets up the target node as node 5 through `node <- c(5)`. Using `igraph::max_cliques`, it extracts all maximal cliques within the graph, considering all possible cliques without specifying minimum or maximum sizes (`min=NULL,max=NULL`). The `subset = node` argument confines the search to cliques containing node 5. The resulting `my_simplified_graph.5clique` holds the identified cliques. These cliques denote subsets of nodes where every pair is directly connected, revealing densely interconnected regions centered around node 5 within the graph.

The largest Cliques of graph:-

- my_simplified_graph.lgcliques=igraph::clique_num(my_simplified_graph)
- my_simplified_graph.lgcliques



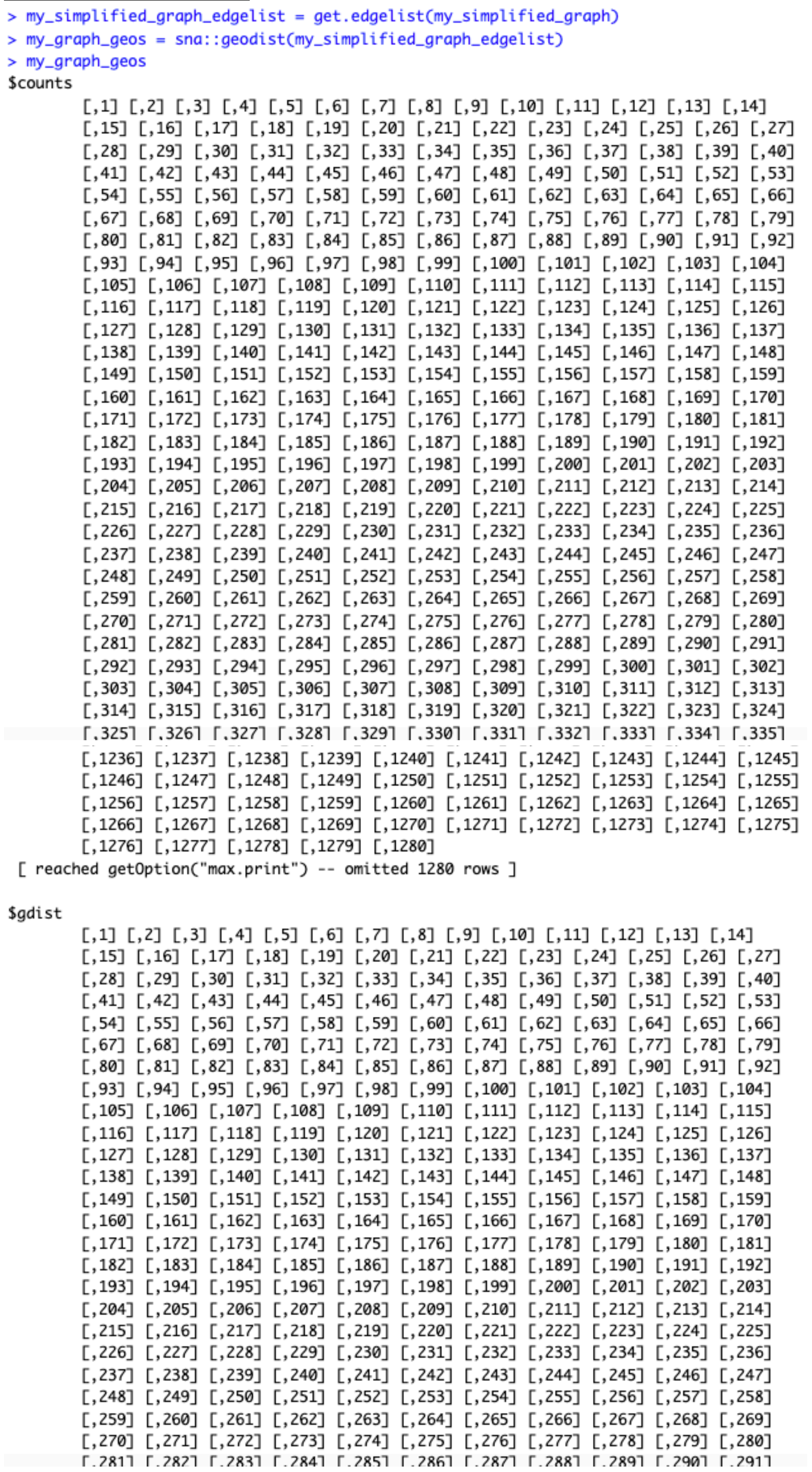
```
> # Calculating the number of cliques in the simplified graph
> my_simplified_graph_lgcliques <- igraph::clique_num(my_simplified_graph)
Warning message:
In igraph::clique_num(my_simplified_graph) :
  At vendor/cigraph/src/cliques/maximal_cliques_template.h:220 : Edge directions are ignored for maximal clique calculation.
> my_simplified_graph_lgcliques
[1] 10
```

Figure 3.16

In Figure 3.16,

The code computes the number of vertices in the largest cliques of the simplified graph. Utilizing `igraph::clique_num`, it determines the size of the largest cliques present in the graph. The result, stored in `my_simplified_graph.lgcliques`, represents the number of vertices within these largest cliques. Larger clique sizes indicate densely connected subsets of nodes within the graph, highlighting significant cohesive structures or communities.

Geodist function:-



```
> my_simplified_graph_edgelist = get.edgelist(my_simplified_graph)
> my_graph_geos = sna::geodist(my_simplified_graph_edgelist)
> my_graph_geos
$counts
        [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
        [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25] [,26] [,27]
        [,28] [,29] [,30] [,31] [,32] [,33] [,34] [,35] [,36] [,37] [,38] [,39] [,40]
        [,41] [,42] [,43] [,44] [,45] [,46] [,47] [,48] [,49] [,50] [,51] [,52] [,53]
        [,54] [,55] [,56] [,57] [,58] [,59] [,60] [,61] [,62] [,63] [,64] [,65] [,66]
        [,67] [,68] [,69] [,70] [,71] [,72] [,73] [,74] [,75] [,76] [,77] [,78] [,79]
        [,80] [,81] [,82] [,83] [,84] [,85] [,86] [,87] [,88] [,89] [,90] [,91] [,92]
        [,93] [,94] [,95] [,96] [,97] [,98] [,99] [,100] [,101] [,102] [,103] [,104]
        [,105] [,106] [,107] [,108] [,109] [,110] [,111] [,112] [,113] [,114] [,115]
        [,116] [,117] [,118] [,119] [,120] [,121] [,122] [,123] [,124] [,125] [,126]
        [,127] [,128] [,129] [,130] [,131] [,132] [,133] [,134] [,135] [,136] [,137]
        [,138] [,139] [,140] [,141] [,142] [,143] [,144] [,145] [,146] [,147] [,148]
        [,149] [,150] [,151] [,152] [,153] [,154] [,155] [,156] [,157] [,158] [,159]
        [,160] [,161] [,162] [,163] [,164] [,165] [,166] [,167] [,168] [,169] [,170]
        [,171] [,172] [,173] [,174] [,175] [,176] [,177] [,178] [,179] [,180] [,181]
        [,182] [,183] [,184] [,185] [,186] [,187] [,188] [,189] [,190] [,191] [,192]
        [,193] [,194] [,195] [,196] [,197] [,198] [,199] [,200] [,201] [,202] [,203]
        [,204] [,205] [,206] [,207] [,208] [,209] [,210] [,211] [,212] [,213] [,214]
        [,215] [,216] [,217] [,218] [,219] [,220] [,221] [,222] [,223] [,224] [,225]
        [,226] [,227] [,228] [,229] [,230] [,231] [,232] [,233] [,234] [,235] [,236]
        [,237] [,238] [,239] [,240] [,241] [,242] [,243] [,244] [,245] [,246] [,247]
        [,248] [,249] [,250] [,251] [,252] [,253] [,254] [,255] [,256] [,257] [,258]
        [,259] [,260] [,261] [,262] [,263] [,264] [,265] [,266] [,267] [,268] [,269]
        [,270] [,271] [,272] [,273] [,274] [,275] [,276] [,277] [,278] [,279] [,280]
        [,281] [,282] [,283] [,284] [,285] [,286] [,287] [,288] [,289] [,290] [,291]
        [,292] [,293] [,294] [,295] [,296] [,297] [,298] [,299] [,300] [,301] [,302]
        [,303] [,304] [,305] [,306] [,307] [,308] [,309] [,310] [,311] [,312] [,313]
        [,314] [,315] [,316] [,317] [,318] [,319] [,320] [,321] [,322] [,323] [,324]
        [,325] [,326] [,327] [,328] [,329] [,330] [,331] [,332] [,333] [,334] [,335]
        [,1236] [,1237] [,1238] [,1239] [,1240] [,1241] [,1242] [,1243] [,1244] [,1245]
        [,1246] [,1247] [,1248] [,1249] [,1250] [,1251] [,1252] [,1253] [,1254] [,1255]
        [,1256] [,1257] [,1258] [,1259] [,1260] [,1261] [,1262] [,1263] [,1264] [,1265]
        [,1266] [,1267] [,1268] [,1269] [,1270] [,1271] [,1272] [,1273] [,1274] [,1275]
        [,1276] [,1277] [,1278] [,1279] [,1280]
 [ reached getOption("max.print") -- omitted 1280 rows ]

$gdist
        [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
        [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25] [,26] [,27]
        [,28] [,29] [,30] [,31] [,32] [,33] [,34] [,35] [,36] [,37] [,38] [,39] [,40]
        [,41] [,42] [,43] [,44] [,45] [,46] [,47] [,48] [,49] [,50] [,51] [,52] [,53]
        [,54] [,55] [,56] [,57] [,58] [,59] [,60] [,61] [,62] [,63] [,64] [,65] [,66]
        [,67] [,68] [,69] [,70] [,71] [,72] [,73] [,74] [,75] [,76] [,77] [,78] [,79]
        [,80] [,81] [,82] [,83] [,84] [,85] [,86] [,87] [,88] [,89] [,90] [,91] [,92]
        [,93] [,94] [,95] [,96] [,97] [,98] [,99] [,100] [,101] [,102] [,103] [,104]
        [,105] [,106] [,107] [,108] [,109] [,110] [,111] [,112] [,113] [,114] [,115]
        [,116] [,117] [,118] [,119] [,120] [,121] [,122] [,123] [,124] [,125] [,126]
        [,127] [,128] [,129] [,130] [,131] [,132] [,133] [,134] [,135] [,136] [,137]
        [,138] [,139] [,140] [,141] [,142] [,143] [,144] [,145] [,146] [,147] [,148]
        [,149] [,150] [,151] [,152] [,153] [,154] [,155] [,156] [,157] [,158] [,159]
        [,160] [,161] [,162] [,163] [,164] [,165] [,166] [,167] [,168] [,169] [,170]
        [,171] [,172] [,173] [,174] [,175] [,176] [,177] [,178] [,179] [,180] [,181]
        [,182] [,183] [,184] [,185] [,186] [,187] [,188] [,189] [,190] [,191] [,192]
        [,193] [,194] [,195] [,196] [,197] [,198] [,199] [,200] [,201] [,202] [,203]
        [,204] [,205] [,206] [,207] [,208] [,209] [,210] [,211] [,212] [,213] [,214]
        [,215] [,216] [,217] [,218] [,219] [,220] [,221] [,222] [,223] [,224] [,225]
        [,226] [,227] [,228] [,229] [,230] [,231] [,232] [,233] [,234] [,235] [,236]
        [,237] [,238] [,239] [,240] [,241] [,242] [,243] [,244] [,245] [,246] [,247]
        [,248] [,249] [,250] [,251] [,252] [,253] [,254] [,255] [,256] [,257] [,258]
        [,259] [,260] [,261] [,262] [,263] [,264] [,265] [,266] [,267] [,268] [,269]
        [,270] [,271] [,272] [,273] [,274] [,275] [,276] [,277] [,278] [,279] [,280]
        [,281] [,282] [,283] [,284] [,285] [,286] [,287] [,288] [,289] [,290] [,291]
```

Figure 3.17

In Figure 3.17,

- my_simplified_graph_edgelist <- get.edgelist(my_simplified_graph)
- my_graph_geos <- sna::geodist(my_simplified_graph_edgelist)

- my_graph_geos

The code first extracts the edge list representation of the simplified graph using `get.edgelist`, storing it as `my_simplified_graph_edgelist`. Then, `sna::geodist` computes the geodesic distances between all pairs of nodes in the graph using the extracted edge list. The resulting `my_graph_geos` holds the geodesic distances, where each element represents the shortest path length between the corresponding pair of nodes. This information provides insights into the overall connectivity and structural properties of the graph, highlighting distances between nodes and facilitating analyses such as network clustering or centrality assessments.

3. c. As an example what do node density and edge density indicate about the problem space?

Two significant metrics that can shed light on the graph analytics problem space are node density and edge density.

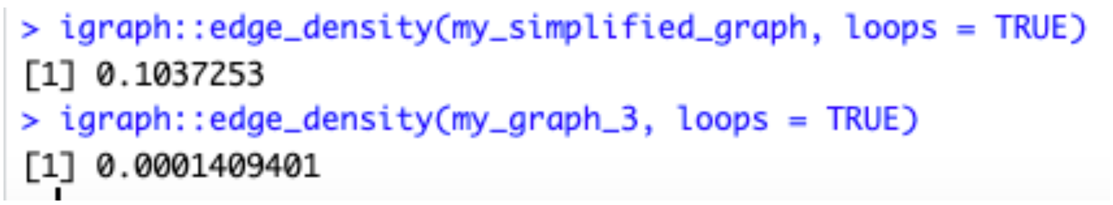
Node Density: Node density is the ratio of the total number of nodes in a graph to the maximum number of nodes. It is calculated as the ratio of the real nodes in the graph to the total number of possible nodes. A large number of nodes in a graph indicates that the problem space has a large number of related objects, which could lead to a complicated and highly connected network. This could make it harder to analyze the relationships between nodes and find important patterns or structures in the data.

Edge Density: The ratio of edges in a graph to all possible edges is known as edge density. It is calculated as the difference between the graph's actual edge count and its maximum possible count. A graph with a high edge density indicates that there are many connections between nodes, which may make it more difficult to analyse the relationships between nodes. A high edge density may also indicate multiple opportunities for information to spread quickly throughout the network, such as influence.

All things considered, node density and edge density provide useful information about the graph analytics problem domain. Understanding these metrics can help researchers and analysts identify structures, linkages, and trends in the data, as well as develop effective methods for efficiently analysing and visualising large, complicated networks.

From the simplified graph we have obtained an edge density of 0.1037 which means that the graph has 10 percent of all the possible edges in the graph. It is sparse but more connected compared to an unsimplified graph.

From the unsimplified graph, we got an edge density of 0.000140 which meant that only 0.014 percent of all possible edges are present in the graph.

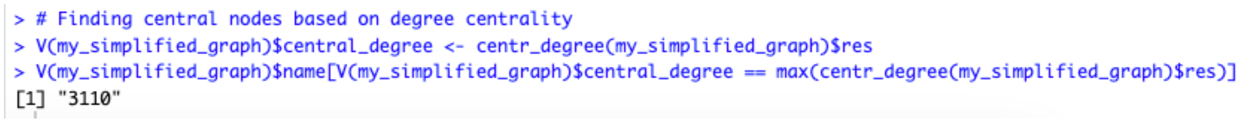


```
> igraph::edge_density(my_simplified_graph, loops = TRUE)
[1] 0.1037253
> igraph::edge_density(my_graph_3, loops = TRUE)
[1] 0.0001409401
```

4. Determine the (a) central nodes(s) in the graph, (b) longest path(s), (c) largest clique(s), (d) ego(s), and € power centrality.

a. Central Nodes:

- V(my_simplified_graph)$central_degree <- centr_degree(my_simplified_graph)$res
- V(my_simplified_graph)$name[V(my_simplified_graph)$central_degree ==max(centr_degree(my_simplified_graph)$res)]



```
> # Finding central nodes based on degree centrality
> V(my_simplified_graph)$central_degree <- centr_degree(my_simplified_graph)$res
> V(my_simplified_graph)$name[V(my_simplified_graph)$central_degree == max(centr_degree(my_simplified_graph)$res)]
[1] "3110"
```

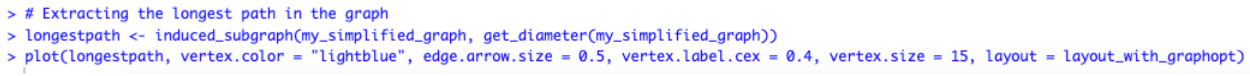
Figure 4.1

In Figure 4.1,

This code identifies central nodes in the simplified graph based on their degree centrality. Firstly, it computes the degree centrality of each node in the graph using **centr_degree**, storing the results in **V(my_simplified_graph)$central_degree**. Then, it selects nodes with the maximum degree centrality by comparing **V(my_simplified_graph)$central_degree** to the maximum value obtained from **centr_degree(my_simplified_graph)$res**. The names of nodes meeting this criterion are extracted using **V(my_simplified_graph)$name[]**. This process highlights nodes with the highest degree centrality, indicating their importance in terms of connectivity within the graph. Such central nodes often play pivotal roles in information flow, communication, or influence propagation within the network.

b. Longest paths:

- longestpath <- induced_subgraph(my_simplified_graph, get_diameter(my_simplified_graph))
- plot(longestpath, vertex.color = "lightblue", edge.arrow.size = 0.5, vertex.label.cex = 0.4, vertex.size = 15, layout = layout_with_graphopt)

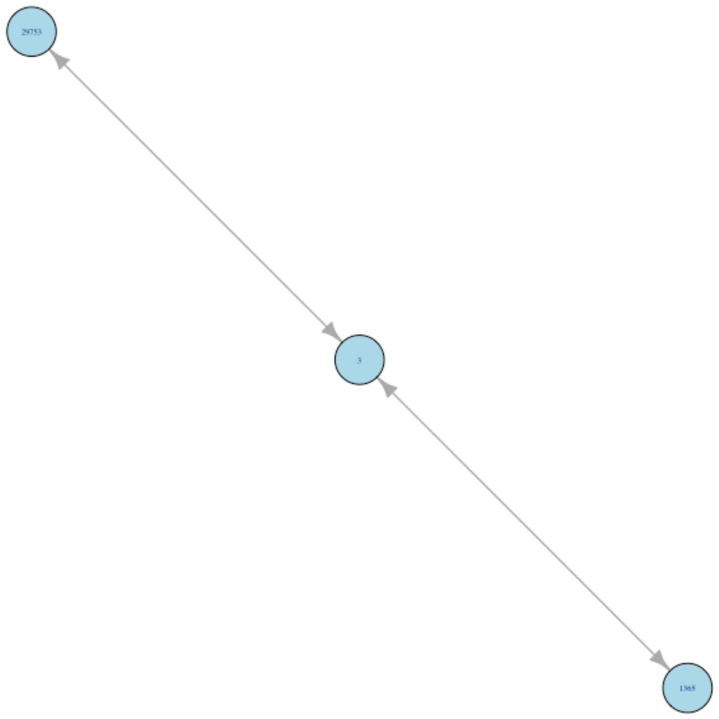


```
> # Extracting the longest path in the graph
> longestpath <- induced_subgraph(my_simplified_graph, get_diameter(my_simplified_graph))
> plot(longestpath, vertex.color = "lightblue", edge.arrow.size = 0.5, vertex.label.cex = 0.4, vertex.size = 15, layout = layout_with_graphopt)
```

Figure 4.2

In Figure 4.2,
This code is dedicated to identifying and visualizing the longest paths in the simplified graph. It starts by extracting the induced subgraph containing the longest paths using **induced_subgraph** and specifying the diameter of the graph as the target length. This subgraph, stored in **longestpath**, effectively represents the longest paths within the original graph. Subsequently, the code utilizes **plot** to visualize this subgraph, customizing various visual aspects such as vertex color, edge arrow size, vertex label size, and layout using parameters like **vertex.color**, **edge.arrow.size**, **vertex.label.cex**, **vertex.size**, and **layout**. The resulting plot provides a clear representation of the longest paths, aiding in the visual inspection of the graph's structural characteristics and potential bottlenecks.

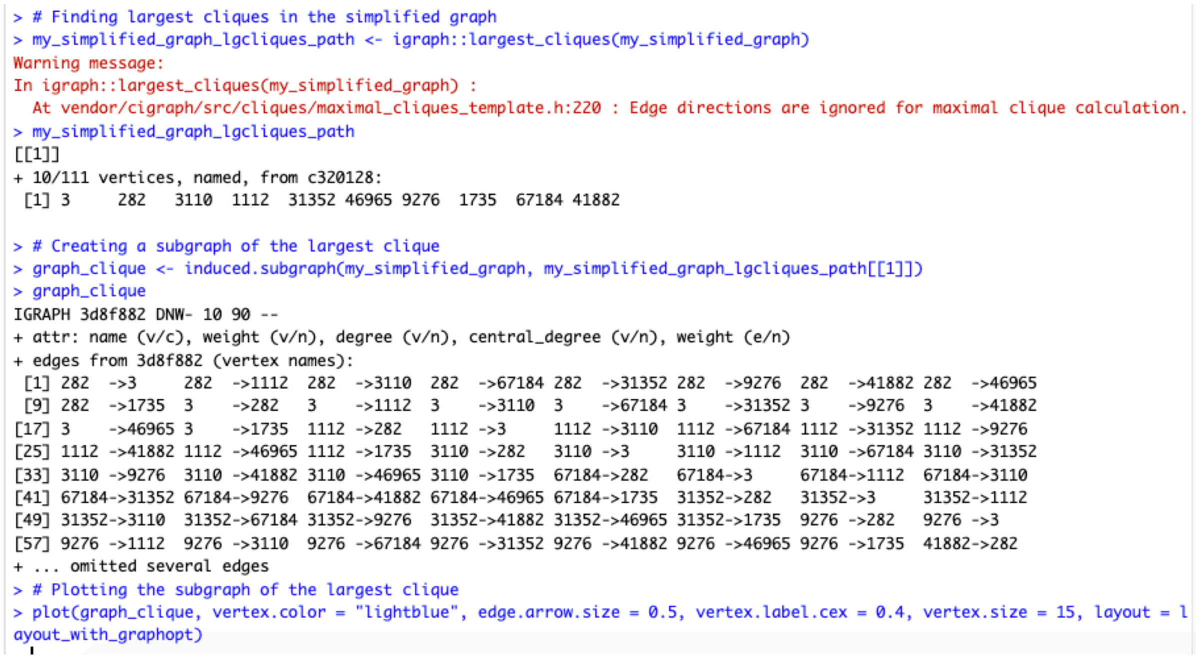
Plot:



c. **Largest Clique:**

- my_simplified_graph.lgcliques=igraph::clique_num(my_simplified_graph)
- my_simplified_graph.lgcliques
- my_simplified_graph.lgcliques_path <- igraph::largest_cliques(my_simplified_graph)
- my_simplified_graph.lgcliques_path
- graph_clique <- induced.subgraph(my_simplified_graph, my_simplified_graph.lgcliques_path[[1]])
- graph_clique

- plot(graph_clique, vertex.color = "lightblue", edge.arrow.size = 0.5, vertex.label.cex = 0.4, vertex.size = 15, layout = layout_with_graphopt)



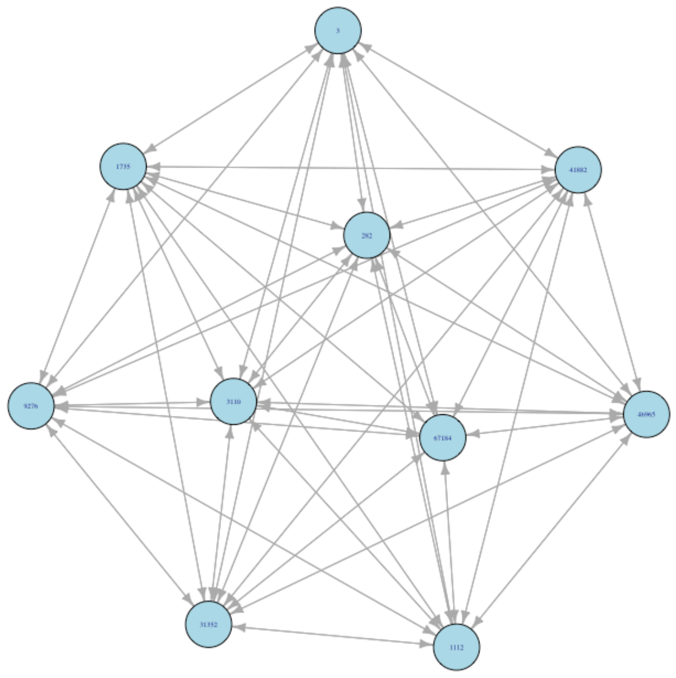
```
> # Finding largest cliques in the simplified graph
> my_simplified_graph_lgcliques_path <- igraph::largest_cliques(my_simplified_graph)
Warning message:
In igraph::largest_cliques(my_simplified_graph) :
  At vendor/cigraph/src/cliques/maximal_cliques_template.h:220 : Edge directions are ignored for maximal clique calculation.
> my_simplified_graph_lgcliques_path
[[1]]
+ 10/111 vertices, named, from c320128:
 [1] 3     282   3110  1112  31352 46965 9276  1735  67184 41882

> # Creating a subgraph of the largest clique
> graph_clique <- induced.subgraph(my_simplified_graph, my_simplified_graph_lgcliques_path[[1]])
> graph_clique
IGRAPH 3d8f882 DNW- 10 90 --
+ attr: name (v/c), weight (v/n), degree (v/n), central_degree (v/n), weight (e/n)
+ edges from 3d8f882 (vertex names):
 [1] 282  ->3     282  ->1112  282  ->3110  282  ->67184 282  ->31352 282  ->9276  282  ->41882 282  ->46965
 [9] 282  ->1735  3    ->282   3    ->1112  3    ->3110  3    ->67184 3    ->31352 3    ->9276  3    ->41882
[17] 3    ->46965 3    ->1735  1112 ->282   1112 ->3     1112 ->3110  1112 ->67184 1112 ->31352 1112 ->9276
[25] 1112 ->41882 1112 ->46965 1112 ->1735  3110 ->282   3110 ->3     3110 ->1112  3110 ->67184 3110 ->31352
[33] 3110 ->9276  3110 ->41882 3110 ->46965 3110 ->1735  67184->282   67184->3     67184->1112  67184->3110
[41] 67184->31352 67184->9276  67184->41882 67184->46965 67184->1735  31352->282   31352->3     31352->1112
[49] 31352->3110  31352->67184 31352->9276  31352->41882 31352->46965 31352->1735  9276 ->282   9276 ->3
[57] 9276 ->1112  9276 ->3110  9276 ->67184 9276 ->31352 9276 ->41882 9276 ->46965 9276 ->1735  41882->282
+ ... omitted several edges
> # Plotting the subgraph of the largest clique
> plot(graph_clique, vertex.color = "lightblue", edge.arrow.size = 0.5, vertex.label.cex = 0.4, vertex.size = 15, layout = l
ayout_with_graphopt)
```

Figure 4.3

In Figure 4.3,

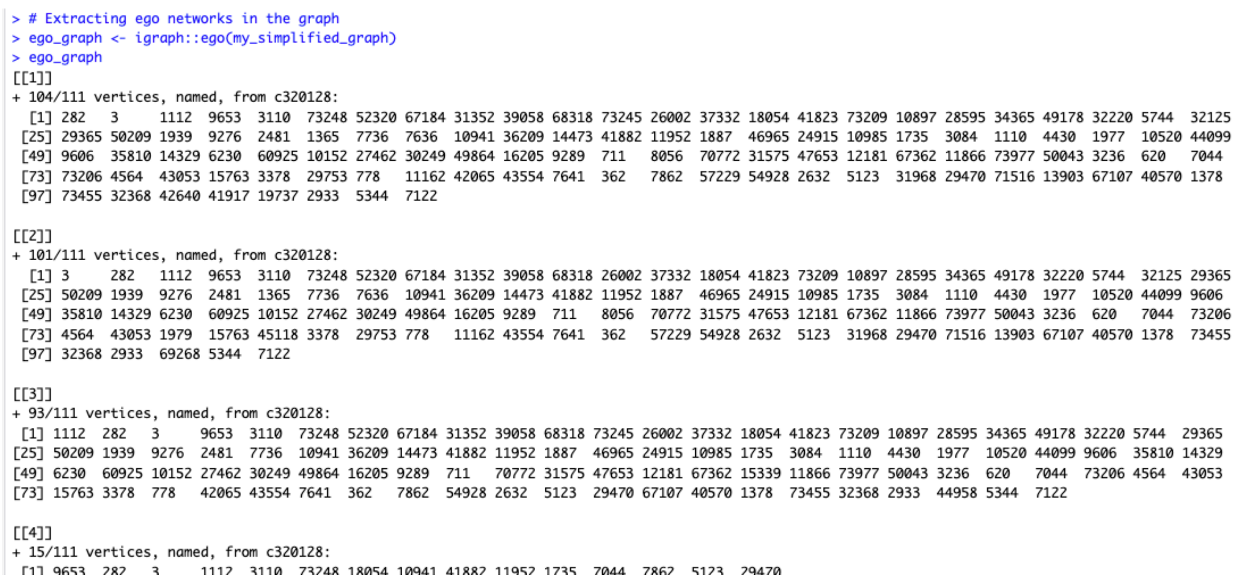
This part of the code focuses on identifying and visualizing the largest clique in the simplified graph. Initially, it computes the size of the largest cliques using `igraph::clique_num`, storing the result in `my_simplified_graph.lgcliques`. Next, it calculates the vertices constituting the largest clique using `igraph::largest_cliques`, storing the result in `my_simplified_graph.lgcliques_path`. Then, it extracts the subgraph corresponding to the largest clique using `induced.subgraph` and the vertices from `my_simplified_graph.lgcliques_path[[1]]`, saved as `graph_clique`. Finally, it plots the largest clique using `plot`, customizing visual attributes like vertex color, edge arrow size, vertex label size, and layout to facilitate a clear representation of the largest clique within the graph. This process offers insights into densely connected subsets of nodes, highlighting cohesive structures within the network.

Plot of largest Clique:



d. Egos:

ego.graph <- igraph::ego(my_simplified_graph)
ego.graph



```
> # Extracting ego networks in the graph
> ego_graph <- igraph::ego(my_simplified_graph)
> ego_graph
[[1]]
+ 104/111 vertices, named, from c320128:
  [1] 282   3     1112  9653  3110  73248 52320 67184 31352 39058 68318 73245 26002 37332 18054 41823 73209 10897 28595 34365 49178 32220 5744  32125
 [25] 29365 50209 1939  9276  2481  1365  7736  7636  10941 36209 14473 41882 11952 1887  46965 24915 10985 1735  3084  1110  4430  1977  10520 44099
 [49] 9606  35810 14329 6230  60925 10152 27462 30249 49864 16205 9289  711   8056  70772 31575 47653 12181 67362 11866 73977 50043 3236  620   7044
 [73] 73206 4564  43053 15763 3378  29753 778   11162 42065 43554 7641  362   7862  57229 54928 2632  5123  31968 29470 71516 13903 67107 40570 1378
 [97] 73455 32368 42640 41917 19737 2933  5344  7122

[[2]]
+ 101/111 vertices, named, from c320128:
  [1] 3     282   1112  9653  3110  73248 52320 67184 31352 39058 68318 26002 37332 18054 41823 73209 10897 28595 34365 49178 32220 5744  32125 29365
 [25] 50209 1939  9276  2481  1365  7736  7636  10941 36209 14473 41882 11952 1887  46965 24915 10985 1735  3084  1110  4430  1977  10520 44099 9606
 [49] 35810 14329 6230  60925 10152 27462 30249 49864 16205 9289  711   8056  70772 31575 47653 12181 67362 11866 73977 50043 3236  620   7044  73206
 [73] 4564  43053 1979  15763 45118 3378  29753 778   11162 43554 7641  362   57229 54928 2632  5123  31968 29470 71516 13903 67107 40570 1378  73455
 [97] 32368 2933  69268 5344  7122

[[3]]
+ 93/111 vertices, named, from c320128:
 [1] 1112  282   3     9653  3110  73248 52320 67184 31352 39058 68318 73245 26002 37332 18054 41823 73209 10897 28595 34365 49178 32220 5744  29365
[25] 50209 1939  9276  2481  7736  10941 36209 14473 41882 11952 1887  46965 24915 10985 1735  3084  1110  4430  1977  10520 44099 9606  35810 14329
[49] 6230  60925 10152 27462 30249 49864 16205 9289  711   70772 31575 47653 12181 67362 15339 11866 73977 50043 3236  620   7044  73206 4564  43053
[73] 15763 3378  778   42065 43554 7641  362   7862  54928 2632  5123  29470 67107 40570 1378  73455 32368 2933  44958 5344  7122

[[4]]
+ 15/111 vertices, named, from c320128:
 [1] 9653  282   3     1112  3110  73248 18054 10941 41882 11952 1735  7044  7862  5123  29470
```

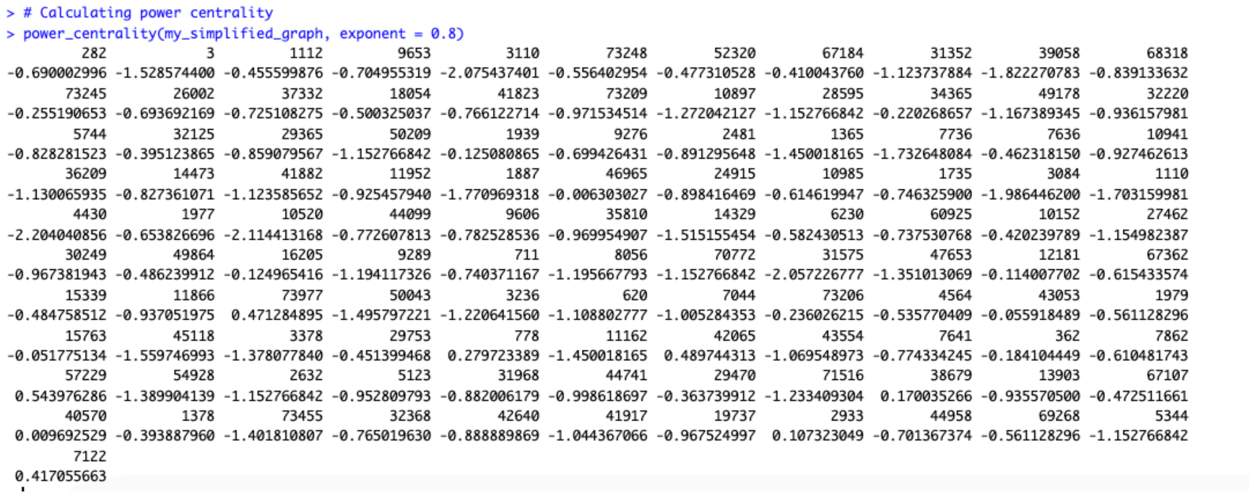
Figure 4.4

In Figure 4.4,

This code calculates and displays the ego networks within the simplified graph. It employs `igraph::ego` to extract the ego networks, which are subgraphs consisting of a focal node and its immediate neighbors. The resulting `ego.graph` contains these ego networks, providing insights into the local network structures surrounding each node. This analysis aids in understanding the relationships and interactions between individual nodes and their immediate neighbors within the broader network context.

e. Power Centrality:

power_centrality(my_simplified_graph, exponent = 0.8)



```
> # Calculating power centrality
> power_centrality(my_simplified_graph, exponent = 0.8)
         282            3         1112         9653         3110        73248        52320        67184        31352        39058        68318 
-0.690002996 -1.528574400 -0.455599876 -0.704955319 -2.075437401 -0.556402954 -0.477310528 -0.410043760 -1.123737884 -1.822270783 -0.839133632 
       73245        26002        37332        18054        41823        73209        10897        28595        34365        49178        32220 
-0.255190653 -0.693692169 -0.725108275 -0.500325037 -0.766122714 -0.971534514 -1.272042127 -1.152766842 -0.220268657 -1.167389345 -0.936157981 
        5744        32125        29365        50209         1939         9276         2481         1365         7736         7636        10941 
-0.828281523 -0.395123865 -0.859079567 -1.152766842 -0.125080865 -0.699426431 -0.891295648 -1.450018165 -1.732648084 -0.462318150 -0.927462613 
       36209        14473        41882        11952         1887        46965        24915        10985         1735         3084         1110 
-1.130065935 -0.827361071 -1.123585652 -0.925457940 -1.770969318 -0.006303027 -0.898416469 -0.614619947 -0.746325900 -1.986446200 -1.703159981 
        4430         1977        10520        44099         9606        35810        14329         6230        60925        10152        27462 
-2.204040856 -0.653826696 -2.114413168 -0.772607813 -0.782528536 -0.969954907 -1.515155454 -0.582430513 -0.737530768 -0.420239789 -1.154982387 
       30249        49864        16205         9289          711         8056        70772        31575        47653        12181        67362 
-0.967381943 -0.486239912 -0.124965416 -1.194117326 -0.740371167 -1.195667793 -1.152766842 -2.057226777 -1.351013069 -0.114007702 -0.615433574 
       15339        11866        73977        50043         3236          620         7044        73206         4564        43053         1979 
-0.484758512 -0.937051975  0.471284895 -1.495797221 -1.220641560 -1.108802777 -1.005284353 -0.236026215 -0.535770409 -0.055918489 -0.561128296 
       15763        45118         3378        29753          778        11162        42065        43554         7641          362         7862 
-0.051775134 -1.559746993 -1.378077840 -0.451399468  0.279723389 -1.450018165  0.489744313 -1.069548973 -0.774334245 -0.184104449 -0.610481743 
       57229        54928         2632         5123        31968        44741        29470        71516        38679        13903        67107 
 0.543976286 -1.389904139 -1.152766842 -0.952809793 -0.882006179 -0.998618697 -0.363739912 -1.233409304  0.170035266 -0.935570500 -0.472511661 
       40570         1378        73455        32368        42640        41917        19737         2933        44958        69268         5344 
 0.009692529 -0.393887960 -1.401810807 -0.765019630 -0.888889869 -1.044367066 -0.967524997  0.107323049 -0.701367374 -0.561128296 -1.152766842 
        7122 
 0.417055663 
```

Figure 4.5

In Figure 4.5,

With an exponent value of 0.8, the algorithm determines the power centrality of each node in the simplified graph. By taking into account the strength of the connections that emanate from nodes, power centrality calculates their influence or relevance inside a network; higher values denote greater centrality. The computation applies a nonlinear adjustment to the edge weights by setting the exponent parameter to 0.8, which may increase the relevance of nodes with stronger connections while decreasing that of nodes with weaker connections. This study provides important insights into the structural and functional characteristics of the network by assisting in the identification of nodes that have the greatest influence on information flow or interaction dynamics.

6. Discuss what you learned from doing this project.
A. From this project, We can learn how to use R and the igraph package to work with huge graph datasets from this project. We specifically learn how to import the soc-Epinions1_adj.tsv dataset into R, use igraph to turn it into a graph object and use the plot function to display the graph.

Additionally, we learn how to use a variety of graph analytics functions on the graph object, such as finding the graph density, utilizing centrality measures to identify significant nodes, and utilizing clustering techniques to find communities within the network. In many different fields, including social network analysis and transportation network analysis, these functions can offer insightful information about the links and structure of the graph, which can be utilized to make well-informed judgments or predictions.

We also discover how crucial node and edge densities are to comprehending the problem space. High edge density indicates that there are many connections between nodes, which can make it more difficult to analyze the relationships between nodes. High node density indicates that there are many interconnected entities in the problem space, which can lead to a more complex and interconnected network. Additionally, we may learn how to visualize the graph and identify key features like power centrality, greatest cliques, longest pathways, and core nodes. Knowing these metrics can aid researchers and analysts in finding structures, linkages, and trends in the data as well as in creating efficient plans for dissecting and displaying intricate and sizable networks.

List of functions used to perform graph operations:

- ⇒ p_load()
- ⇒ read.table()
- ⇒ head()
- ⇒ nrow()
- ⇒ graph_from_data_frame()
- ⇒ plot()
- ⇒ plot.igraph()
- ⇒ cluster_walktrap()
- ⇒ contract()
- ⇒ E()
- ⇒ V()
- ⇒ simplify()
- ⇒ induced_subgraph()
- ⇒ delete_vertices()

- ⇒ as_adjacency_matrix ()
- ⇒ get_edgelist ()
- ⇒ gden()
- ⇒ edge_density()
- ⇒ sna::ego.extract()
- ⇒ sna::geodist()
- ⇒ igraph::degree()
- ⇒ igraph::centr_betw()
- ⇒ igraph::centr_clo()
- ⇒ igraph::shortest_paths()
- ⇒ igraph::distances()
- ⇒ igraph::diameter()
- ⇒ igraph::max_cliques()
- ⇒ igraph::clique_num()
- ⇒ igraph::largest_cliques()
- ⇒ centr_degree()
- ⇒ get_diameter()
- ⇒ igraph::ego()
- ⇒ power_centrality()

THANK YOU