Practical No:01

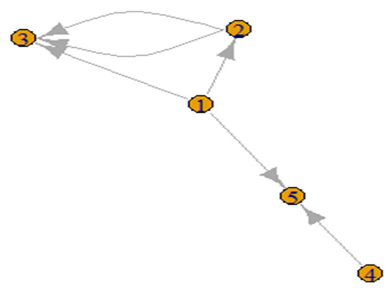
Aim: Write a program to compute the following for a given a network: (i) number of edges, (ii) number of nodes; (iii) degree of node; (iv) node with lowest degree; (v)the adjacency list; (vi) matrix of the graph.

(i) number of edges

```
library(igraph)
edges <- c(1, 2, 1, 3, 2, 3, 2, 3, 4, 5, 1, 5)
g <- graph(edges, directed = TRUE)
plot(g)
```

Output:-

```
> library(igraph)
> edges <- c(1, 2, 1, 3, 2, 3, 2, 3, 4, 5, 1, 5)
> g <- graph(edges, directed = TRUE)
```




```
degree(g)
E(g)
v(g)
```

Output:-

```
> degree(g)
[1] 3 3 3 1 2
> E(g)
+ 6/6 edges from 917ebf4:
[1] 1->2 1->3 2->3 2->3 4->5 1->5
> v(g)
```

```
degree(g, mode = "in")
degree(g, mode = "out")
```

Output:-

```
> degree(g, mode = "in")
[1] 0 1 3 0 2
> degree(g, mode = "out")
[1] 3 2 0 1 0
```

```
get.adjacency(g)
```

Output:-

```
> get.adjacency(g)
5 x 5 sparse Matrix of class "dgCMatrix"

[1,] . 1 1 . 1
[2,] . . 2 . .
[3,] . . . . .
[4,] . . . . 1
[5,] . . . . .
```

```
get.adjedgelist(g,mode=c("all"))
```

Output-

```
> get.adjedgelist(g,mode=c("all"))
[[1]]
+ 3/6 edges from 917ebf4:
[1] 1->2 1->3 1->5

[[2]]
+ 3/6 edges from 917ebf4:
[1] 1->2 2->3 2->3

[[3]]
+ 3/6 edges from 917ebf4:
[1] 1->3 2->3 2->3

[[4]]
+ 1/6 edge from 917ebf4:
[1] 4->5

[[5]]
+ 2/6 edges from 917ebf4:
[1] 1->5 4->5
```

Practical No:02

Aim: Perform following tasks: (i) View data collection forms and/or import onemode/ two-mode datasets; (ii) Basic Networks matrices transformations

Id.csv

	A	B	C	D
1	id	media	madia typ	audience size
2	S01	NY times	Newspape	20
3	S02	LA times	Newspape	54
4	S03	CNN	TV	55
5	S04	ABC	TV	24
6	S05	YAHOO	TV	52
7	S06	GOOGLE	TV	85
8	S07	NY times	TV	95
9	S08	WAHINGT	Newspape	65
10	S09	WALL STRI	Newspape	82
11	S10	USA TODA	Newspape	56
12	S11	HINDUSTA	Newspape	24
13	S12	TIMES OF	Newspape	13
14	S13	AOL.com	Newspape	85
15	S14	Facebook	Social Net	13
16	S15	MSNBC	Social Net	63
17	S16	Fox news	TV	45
18	S17	NY times	TV	21
19	S18	WASHING	Newspape	87
20	S19	WASHING	Newspape	29

Weight.csv

	A	B	C
1	from	to	weight
2	S03	S01	24
3	S04	S03	16
4	S05	S02	24
5	S06	S15	26
6	S07	S03	15
7	S08	S15	28
8	S09	S03	18
9	S10	S15	16
10	S11	S02	29
11	S12	S06	15
12	S13	S10	17
13	S14	S04	19
14	S15	S10	25
15	S16	S03	24
16	S17	S15	26
17	S18	S10	15
18	S19	S04	25

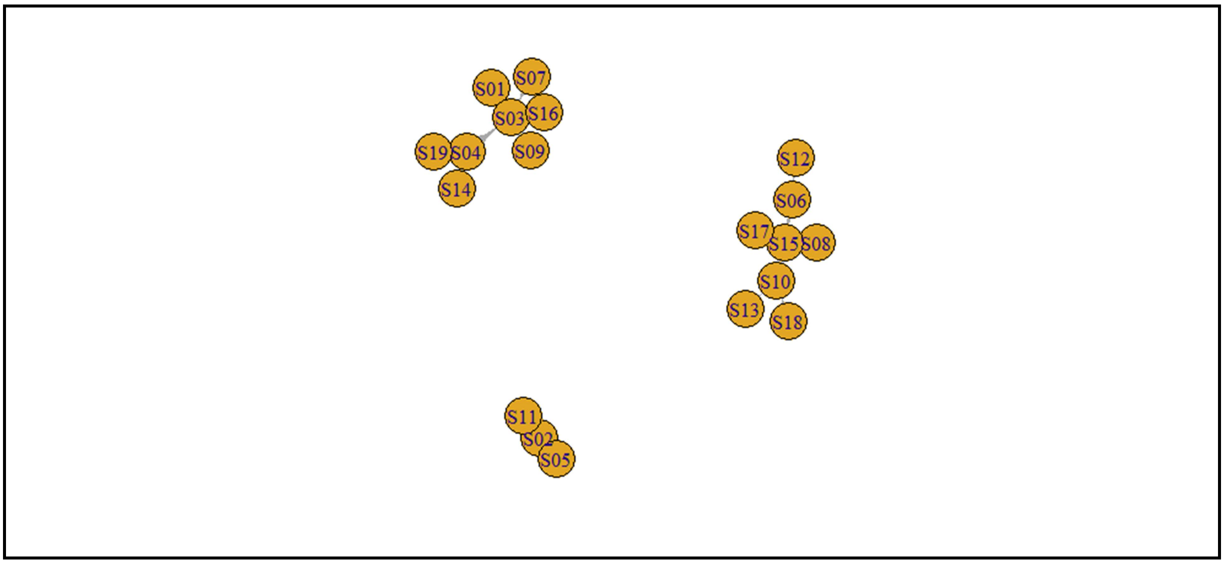
R studio package install

car,cardata,igraph,lme4,readxl,rjava,xlsx,xlsxjars

Commands

```
> nodes <- read.csv("C:/Users/DELL/Desktop/id.csv")
> head(nodes)
   id     media madia.type audience.size
1 S01  NY times  Newspaper            20
2 S02  LA times  Newspaper            54
3 S03       CNN         TV            55
4 S04       ABC         TV            24
5 S05     YAHOO         TV            52
6 S06    GOOGLE         TV            85
```

```
> links <- read.csv("C:/Users/DELL/Desktop/weight.csv")
> head(links)
  ï..from  to weight
1     S03 S01     24
2     S04 S03     16
3     S05 S02     24
4     S06 S15     26
5     S07 S03     15
6     S08 S15     28
> ntgrp<-graph_from_data_frame(d=links,v=nodes1,directed = T)
> plot(ntgrp)
> plot(ntgrp,vertex.size=20)
```

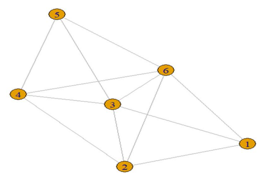


Practical No:03

Aim: Compute the following node level measures: (i) Density; (ii) Degree;(iii) Reciprocity; (iv) Transitivity; (v) Centralization; (vi) Clustering

```
g<-graph.formula(1-2,1-3,2-4,2-3,4-2,2-4,3-4,4-5,5-3,5-6,6-1,3-6,6-4,6-2)
plot(g)
```

Output:



1. Density

```
ecount(g)/(vcount(g)*vcount(g)-1)
```

Output:

```
> ecount(g)/(vcount(g)*vcount(g)-1)
[1] 0.3428571
```

2. Degree

```
degree(g)
ecount(g)
vcount(g)
```

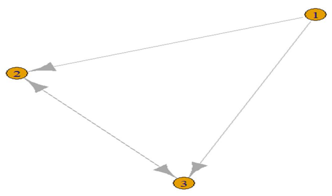
Output:

```
> degree(g)
1 2 3 4 5 6
3 4 5 4 3 5
> ecount(g)
[1] 12
> vcount(g)
[1] 6
```

3.Reciprocity

```
library(igraph)
g<-graph.formula(1-+2,1-+3,2++3)
plot(g)
reciprocity(g)
dyad.census(g)
```

Output:

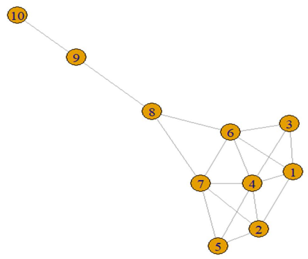



```
> reciprocity(g)
[1] 0.5
> dyad.census(g)
$mut
[1] 1

$asym
[1] 2

$null
[1] 0
```

4.transitivity

```
kite<-graph.famous("krackhardt_kite")
net<-adjacent.triangles(kite)
plot (kite,vertex.lable=net)
transitivity(kite,type="local")
```

Output:



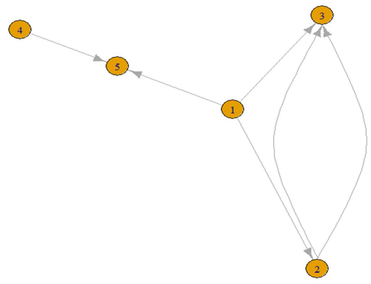

```
> transitivity(kite,type="local")
 [1] 0.6666667 0.6666667 1.0000000 0.5333333 1.0000000 0.5000000 0.5000000
 [8] 0.3333333 0.0000000       NaN
```

5.Centrality

```
library(igraph)
edges <- c(1, 2, 1, 3, 2, 3, 2, 3, 4, 5, 1, 5)
g <- graph(edges, directed = TRUE)
plot(g)
centralization.degree(g, mode="out", normalized = T)
centralization.degree(g, mode="in", normalized = T)
centralization.degree(g, mode="all", normalized = T)
```

Output:-

```
> centralization.degree(g, mode="out", normalized = T)
$res
[1] 3 2 0 1 0

$centralization
[1] 0.45

$theoretical_max
[1] 20

> centralization.degree(g, mode="in", normalized = T)
$res
[1] 0 1 3 0 2

$centralization
[1] 0.45

$theoretical_max
[1] 20

> centralization.degree(g, mode="all", normalized = T)
$res
[1] 3 3 3 1 2

$centralization
[1] 0.09375

$theoretical_max
[1] 32
```



6.Closeness

closeness(g, mode="all")
centralization.closeness(g, mode="all", normalized = T)

Output:

```
> closeness(g, mode="all")
[1] 0.2000000 0.1428571 0.1428571 0.1111111 0.1666667

> centralization.closeness(g, mode="all", normalized = T)
$res
[1] 0.8000000 0.5714286 0.5714286 0.4444444 0.6666667

$centralization
[1] 0.5518519

$theoretical_max
[1] 1.714286
```

7.Betweenness

betweenness(g, directed = T, weights=NA)
edge.betweenness(g, directed = T, weights=NA)
centralization.betweenness(g, directed = T, normalized = NA)

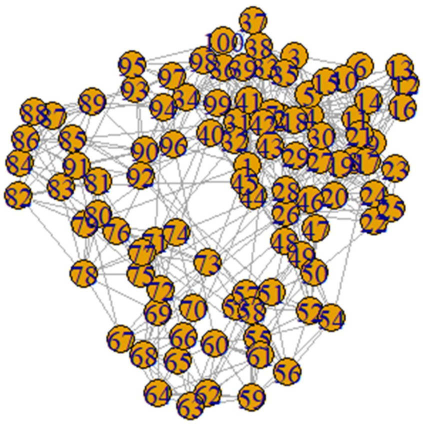
Output:-

```
> betweenness(g, directed = T, weights=NA)
[1] 0 0 0 0 0

> edge.betweenness(g, directed = T, weights=NA)
[1] 1.0 1.0 0.5 0.5 1.0 1.0

> centralization.betweenness(g, directed = T, normalized = NA)
$res
[1] 0 0 0 0 0
```

```
$centralization
[1] 0

$theoretical_max
[1] 48
```

8.Clustering

```
library(igraph)
g2 <- barabasi.game(50, p = 2, directed = FALSE)
g1 <- watts.strogatz.game(1, size = 100, nei = 5, p = 0.05)
g <- graph.union(g1, g2)
g <- simplify(g)
summary(g)
plot(g)
```

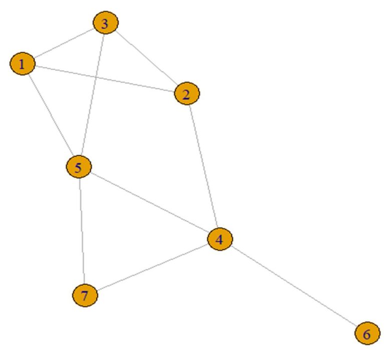


```
> summary(g)
IGRAPH c254979 U--- 100 540 --
+ attr: name_1 (g/c), name_2 (g/c), dim (g/n), size (g/n), nei
| (g/n), p (g/n), loops (g/l), multiple (g/l), power (g/n), m
| (g/n), zero.appeal (g/n), algorithm (g/c)
```

Practical No:04

Aim: For a given network find the following: (i) Length of the shortest path from a given node to another node; (ii) the density of the graph; (iii) Draw egocentric network of node G with chosen configuration parameters.

```
library(igraph)
edges <- c(1, 2, 2, 3, 1, 3, 2, 4, 3, 5, 4, 5, 4, 6, 4, 7, 5, 7, 5, 1)
g <- graph(edges, directed = FALSE)
plot(g)
```



1) Length of the shortest path from a given node to another node

I

```
s <- shortest_paths(g, from = 3, to = 2)
cat("Shortest path from node 3 to 6: ", length(s$v[[1]]) - 1, "\n")
```

```
> cat("Shortest path from node 3 to 6: ", length(s$v[[1]]) - 1, "\n")
Shortest path from node 3 to 6:  1
```

II

```
s <- shortest_paths(g, from = 2, to = 7)
cat("Shortest path from node 2 to 7: ", length(s$v[[1]]) - 1, "\n")
```

```
> cat("Shortest path from node 2 to 7: ", length(s$v[[1]]) - 1, "\n")
Shortest path from node 2 to 7:  2
```

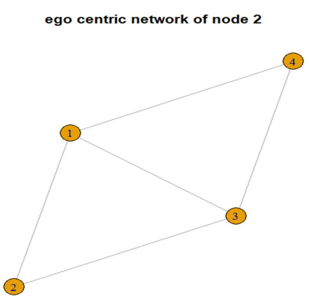
2) The density of the graph

```
density <-edge_density(g)
cat ("density of a graph:",density)
```

```
> cat ("density of a graph:",density)
density of a graph: 0.4761905
```

3) Draw egocentric network of node G with chosen configuration parameters.

```
ego_graph <- make_ego_graph(g,order = 1,nodes  = 1)[[1]]
plot (ego_graph,main = "ego centric network of node 2")
```

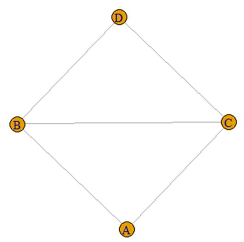


Practical No:05

Aim: Write a program to distinguish between a network as a matrix, a network as an edge list, and the network as a sociogram (or “network graph”) using 3 distinct network representatives of each.

```
library(igraph)

# 1. Network as an Adjacency Matrix
adj_matrix1 <- matrix(c(0, 1, 1, 0,
                        1, 0, 1, 1,
                        1, 1, 0, 1,
                        0, 1, 1, 0),
                      nrow = 4, byrow = TRUE)
colnames(adj_matrix1) <- rownames(adj_matrix1) <- c("A", "B", "C", "D")
print("Adjacency Matrix Representation (Network 1):")
print(adj_matrix1)

graph_matrix1 <- graph_from_adjacency_matrix(adj_matrix1, mode = "undirected")
plot(graph_matrix1, main = "Network 1 as Adjacency Matrix")
```

Network 1 as Adjacency Matrix



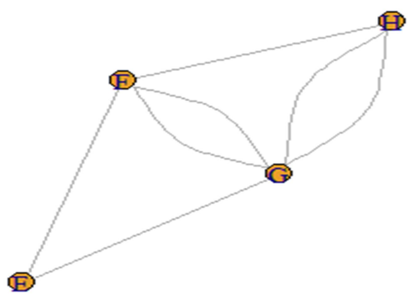

```
> library(igraph)
>
> # 1. Network as an Adjacency Matrix
> adj_matrix1 <- matrix(c(0, 1, 1, 0,
+                         1, 0, 1, 1,
+                         1, 1, 0, 1,
+                         0, 1, 1, 0),
+                       nrow = 4, byrow = TRUE)
> colnames(adj_matrix1) <- rownames(adj_matrix1) <- c("A", "B", "C", "D")
> print("Adjacency Matrix Representation (Network 1):")
```

```
[1] "Adjacency Matrix Representation (Network 1):"
> print(adj_matrix1)
  A B C D
A 0 1 1 0
B 1 0 1 1
C 1 1 0 1
D 0 1 1 0
>
> graph_matrix1 <- graph_from_adjacency_matrix(adj_matrix1, mode =
"undirected")
> plot(graph_matrix1, main = "Network 1 as Adjacency Matrix")
```

```
# 2. Network as an Edge List
edge_list2 <- data.frame(from = c("E", "E", "F", "F", "G", "G", "H"),
                         to = c("F", "G", "G", "H", "H", "F", "G"))
print("Edge List Representation (Network 2):")
print(edge_list2)

graph_edge_list2 <- graph_from_data_frame(edge_list2, directed = FALSE)
plot(graph_edge_list2, main = "Network 2 as Edge List")
```

```
# 2. Network as an Edge List
> edge_list2 <- data.frame(from = c("E", "E", "F", "F", "G", "G", "H"),
+                          to = c("F", "G", "G", "H", "H", "F", "G"))
> print("Edge List Representation (Network 2):")
[1] "Edge List Representation (Network 2):"
> print(edge_list2)
  from to
1    E  F
2    E  G
3    F  G
4    F  H
5    G  H
6    G  F
7    H  G
>
> graph_edge_list2 <- graph_from_data_frame(edge_list2, directed = FALSE)
> plot(graph_edge_list2, main = "Network 2 as Edge List")
```

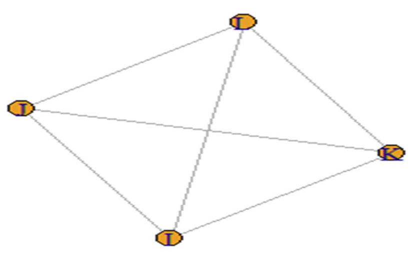
Network 2 as Edge List



```
# 3. Network as a Sociogram (Graph Representation)
nodes3 <- data.frame(name = c("I", "J", "K", "L"))
relations3 <- data.frame(from = c("I", "I", "J", "J", "K", "L"),
                         to = c("J", "K", "K", "L", "L", "I"))
graph_sociogram3 <- graph_from_data_frame(relations3, vertices = nodes3, directed =
FALSE)
plot(graph_sociogram3, main = "Network 3 as Sociogram")
```

```
> # 3. Network as a Sociogram (Graph Representation)
> nodes3 <- data.frame(name = c("I", "J", "K", "L"))
> relations3 <- data.frame(from = c("I", "I", "J", "J", "K", "L"),
+                          to = c("J", "K", "K", "L", "L", "I"))
> graph_sociogram3 <- graph_from_data_frame(relations3, vertices = nodes3,
directed = FALSE)
> plot(graph_sociogram3, main = "Network 3 as Sociogram")
```

Network 3 as Sociogram



Practical No:06

Aim: Write a program to exhibit structural equivalence, automatic equivalence, and regular equivalence from a network.

Code

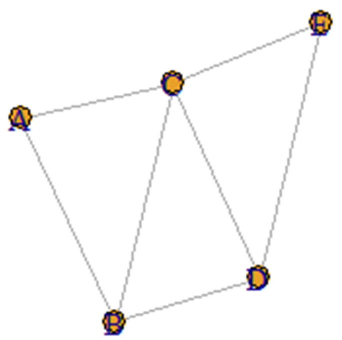
```
library(igraph)
eq_matrix <- matrix(c(0, 1, 1, 0, 0,
                     1, 0, 1, 1, 0,
                     1, 1, 0, 1, 1,
                     0, 1, 1, 0, 1,
                     0, 0, 1, 1, 0),
                   nrow = 5, byrow = TRUE)

colnames(eq_matrix) <- rownames(eq_matrix) <- c("A", "B", "C", "D", "E")

graph_eq <- graph_from_adjacency_matrix(eq_matrix, mode = "undirected")

plot(graph_eq, main = "Network for Equivalence Analysis")
structural_eq <- similarity(graph_eq, method = "jaccard")
print("Structural Equivalence:")
print(structural_eq)

automorphic_eq <- distances(graph_eq)
print("Automorphic Equivalence:")
print(automorphic_eq)
print("Regular Equivalence:")
```

Output-

```
> eq_matrix <- matrix(c(0, 1, 1, 0, 0,
+                      1, 0, 1, 1, 0,
+                      1, 1, 0, 1, 1,
+                      0, 1, 1, 0, 1,
+                      0, 0, 1, 1, 0),
+                    nrow = 5, byrow = TRUE)
>
> colnames(eq_matrix) <- rownames(eq_matrix) <- c("A", "B", "C", "D", "E")
>
> graph_eq <- graph_from_adjacency_matrix(eq_matrix, mode = "undirected")
>
> plot(graph_eq, main = "Network for Equivalence Analysis")
>
>
> structural_eq <- similarity(graph_eq, method = "jaccard")
> print("Structural Equivalence:")
```

```
[1] "Structural Equivalence:"
> print(structural_eq)
          [,1]      [,2] [,3]      [,4]      [,5]
[1,] 1.0000000 0.2500000  0.2 0.6666667 0.3333333
[2,] 0.2500000 1.0000000  0.4 0.2000000 0.6666667
[3,] 0.2000000 0.4000000  1.0 0.4000000 0.2000000
[4,] 0.6666667 0.2000000  0.4 1.0000000 0.2500000
[5,] 0.3333333 0.6666667  0.2 0.2500000 1.0000000
>
>
> automorphic_eq <- distances(graph_eq)
> print("Automorphic Equivalence:")
[1] "Automorphic Equivalence:"
> print(automorphic_eq)
  A B C D E
A 0 1 1 2 2
B 1 0 1 1 2
C 1 1 0 1 1
D 2 1 1 0 1
E 2 2 1 1 0
>
>
> print("Regular Equivalence:")
[1] "Regular Equivalence:"
```

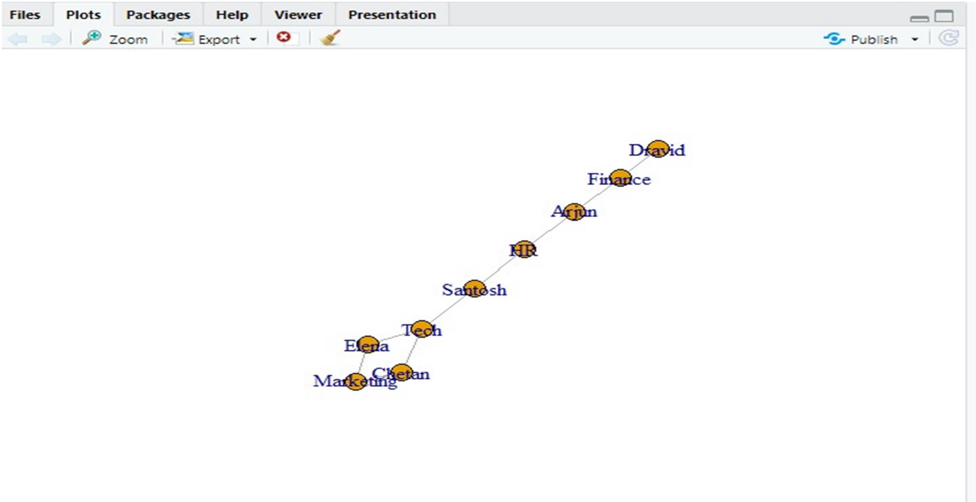
Network for Equivalence Analysis



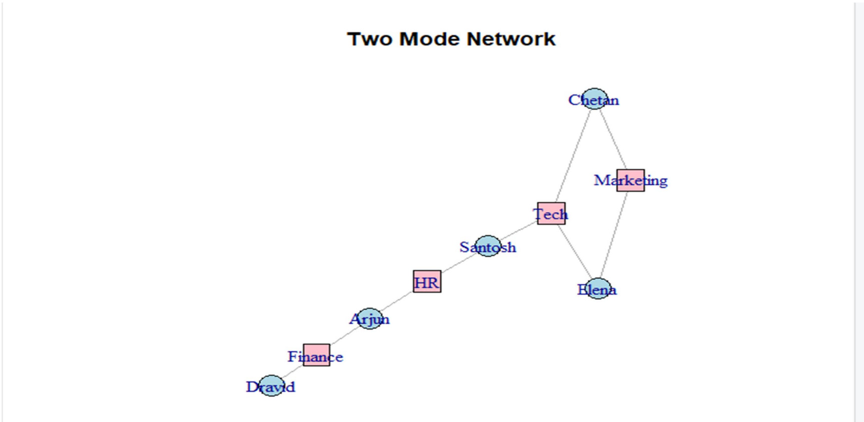
Practical No:07

Aim: Create sociograms for the persons-by-persons network and the committee-by committee network for a given relevant problem. Create one-mode network and two-node network for the same

```
> # Plot the one-mode committee network
> plot(committees_network,
+      vertex.color = "pink",
+      vertex.size = 50,
+      main = "One Mode Committee by Committee Network")
> # Create a one-mode projection of the graph, focusing on persons
> persons_network <- bipartite_projection(g, which = "true")
>
> # Plot the one-mode persons network
> plot(persons_network,
+      vertex.color = "lightblue",
+      vertex.size = 50,
+      main = "One Mode Persons by Persons Network")
> # Check if the committees_network object was created correctly
> print(committees_network)
IGRAPH aab390b UNW- 4 3 --
+ attr: name (v/c), weight (e/n)
+ edges from aab390b (vertex names):
[1] Finance--HR      HR     --Tech      Tech   --Marketing
>
> # Plot the one-mode committee network
> plot(committees_network,
+      vertex.color = "pink",
+      vertex.size = 50,
+      main = "One Mode Committee by Committee Network")
> # Plot the initial graph
> plot(g)
```

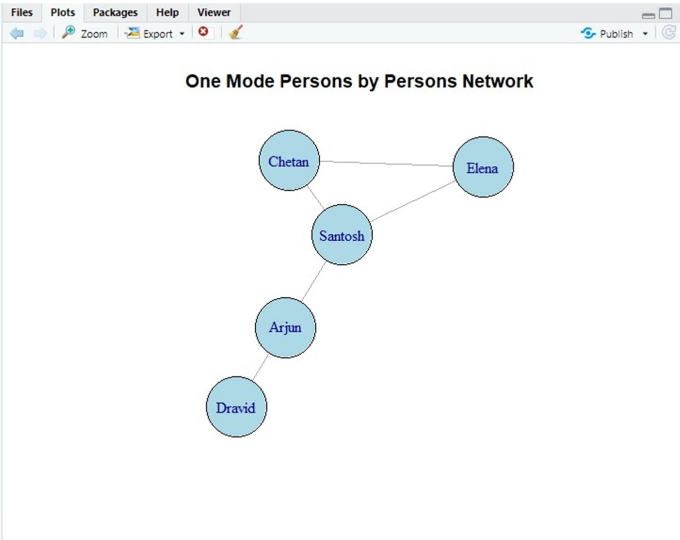



```
# Plot the graph with custom colors and shapes for persons and committees
> plot(g,
+     vertex.color = ifelse(V(g)$type, "lightblue", "pink"),
+     vertex.shape = ifelse(V(g)$type, "circle", "square"),
+     vertex.size = 15,
+     vertex.label.cex = 1,
+     main = "Two Mode Network")
```



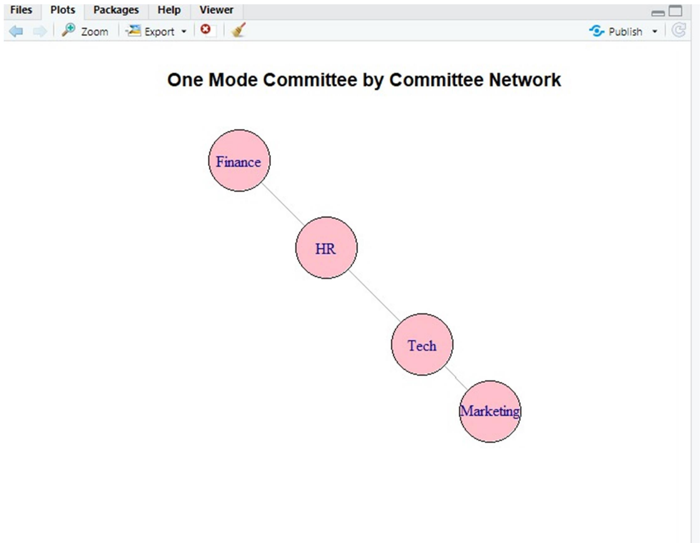

```
> # Check if the persons_network object was created successfully
> print(persons_network)
```

```
IGRAPH 0cd8be6 UNW- 5 5 --
+ attr: name (v/c), weight (e/n)
+ edges from 0cd8be6 (vertex names):
[1] Arjun  --Dravid  Arjun  --Santosh Santosh--Chetan  Santosh--Elena   Chetan --Elena
>
> # Plot the one-mode persons network
> plot(persons_network,
+      vertex.color = "lightblue",
+      vertex.size = 50,
+      main = "One Mode Persons by Persons Network")
```




```
 # Check if the committees_network object was created correctly
> print(committees_network)
IGRAPH aab390b UNW- 4 3 --
+ attr: name (v/c), weight (e/n)
+ edges from aab390b (vertex names):
[1] Finance--HR       HR      --Tech      Tech   --Marketing
>
> # Plot the one-mode committee network
> plot(committees_network,
```

```
+      vertex.color = "pink",
+      vertex.size = 50,
+      main = "One Mode Committee by Committee Network")
```



Practical No:08

Aim: Perform SVD analysis of a network.

```
library(igraph)
g <- graph(c(1, 2, 2, 3, 3, 4, 4, 1))
adj_matrix <- as_adjacency_matrix(g, sparse = FALSE)
print(adj_matrix)
```

```
     [,1] [,2] [,3] [,4]
[1,]    0    1    0    0
[2,]    0    0    1    0
[3,]    0    0    0    1
[4,]    1    0    0    0
```

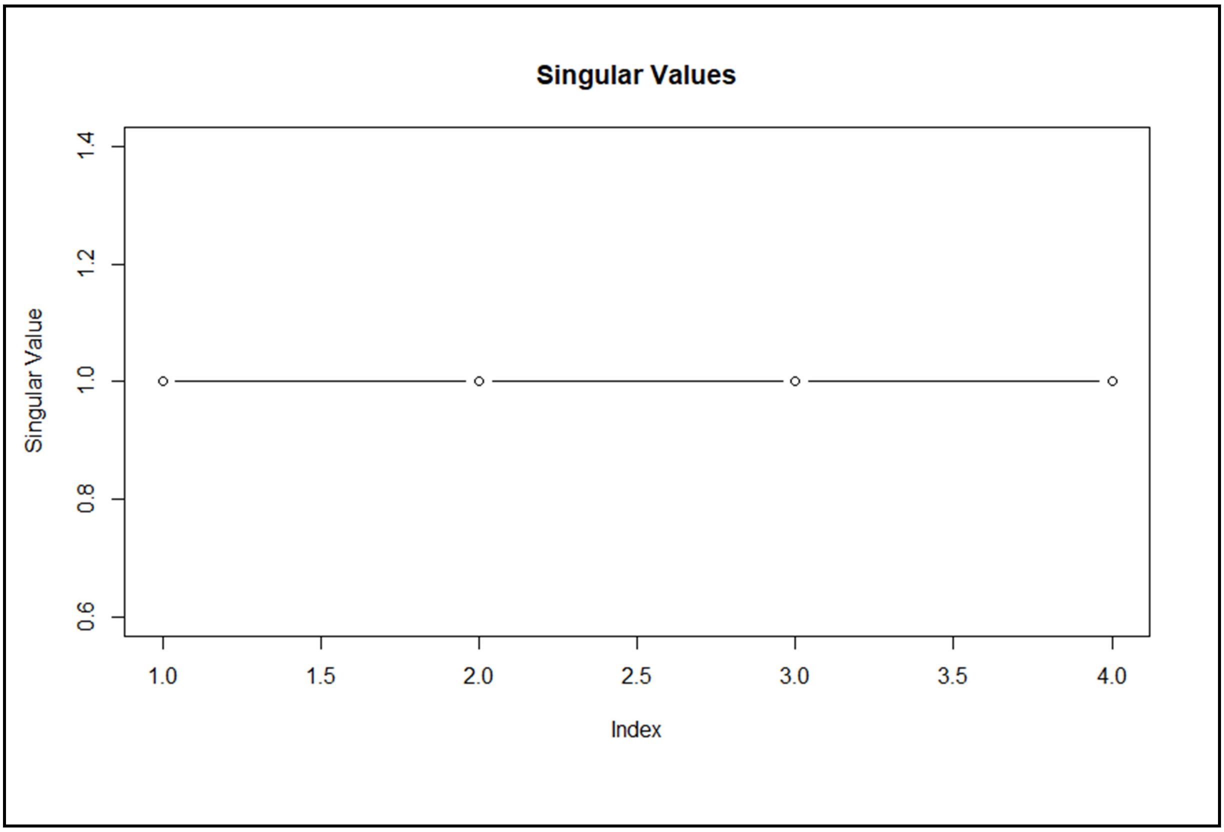
```
svd_result <- svd(adj_matrix)
print(svd_result)
```

```
$d
[1] 1 1 1 1
$u
     [,1] [,2] [,3] [,4]
[1,]    0   -1    0    0
[2,]    0    0   -1    0
[3,]    0    0    0   -1
[4,]   -1    0    0    0
$v
     [,1] [,2] [,3] [,4]
[1,]   -1    0    0    0
[2,]    0   -1    0    0
[3,]    0    0   -1    0
[4,]    0    0    0   -1
```

```
U <- svd_result$u
```

```
D <- diag(svd_result$d)  # D is a diagonal matrix, created with diag() function
V <- svd_result$v
print("U Matrix:")
print(U)
print("D Matrix:")
print(D)
print("V Matrix:")
print(V)

> print(U)
     [,1] [,2] [,3] [,4]
[1,]    0   -1    0    0
[2,]    0    0   -1    0
[3,]    0    0    0   -1
[4,]   -1    0    0    0
>
> print("D Matrix:")
[1] "D Matrix:"
> print(D)
     [,1] [,2] [,3] [,4]
[1,]    1    0    0    0
[2,]    0    1    0    0
[3,]    0    0    1    0
[4,]    0    0    0    1
>
> print("V Matrix:")
[1] "V Matrix:"
> print(V)
     [,1] [,2] [,3] [,4]
[1,]   -1    0    0    0
[2,]    0   -1    0    0
[3,]    0    0   -1    0
[4,]    0    0    0   -1
```

```
plot(svd_result$d, type = "b", main = "Singular Values", xlab = "Index", ylab = "Singular Value")
```


Singular Values
Singular Value
Index
0.6
0.8
1.0
1.2
1.4
1.0
1.5
2.0
2.5
3.0
3.5
4.0

Practical No:09

Aim: Identify ties within the network using two-mode core periphery analysis.

Definitions
Bipartite Graph: A bipartite graph is a graph in which the set of vertices can be divided into two disjoint sets such that no two vertices within the same set are adjacent. This means connections (edges) only occur between nodes of different sets. For example, one set can represent people, and the other set can represent events, with edges indicating which people attended which events.

Periphery: In the context of network analysis, the periphery of a network refers to the nodes that are less connected compared to others. These nodes typically have a lower degree (fewer connections) and are on the outer edges of the network structure. In contrast, core nodes are highly connected and often central to the network.

Steps
Create Bipartite Graph: The bipartite graph is created from the biadjacency matrix using the graph_from_biadjacency_matrix function from the igraph library.
Plot the Bipartite Graph: The bipartite graph is plotted with a title "Bipartite Network" to visualize the network structure.
Calculate Core-Periphery Structure:A simple heuristic based on node degree is used to classify nodes into core and periphery nodes.
Nodes with a degree greater than 2 are considered core nodes, while the rest are considered periphery nodes.
Print Core-Periphery Structure: The core-periphery structure is printed to the console, showing which nodes belong to the core and which belong to the periphery.

Code:

```
library(bipartite)
library(igraph)
#Example of bipartite adjacency matrix (1 indicates a tie between the two actors)
adj_matrix <- matrix(c(
1, 1, 1, 0,
1, 1, 0, 1,
1, 0, 1, 1,
```

```
0, 1, 1, 1),
nrow 4, ncol 4, byrow = TRUE)
rownames(adj_matrix) <- c("Person1", "Person2", "Person3", "Person4")
colnames(adj_matrix) <- c("Event1", "Event2", "Event3", "Event4")
print(adj_matrix)

rownames(adj_matrix) <- c("Person1", "Person2", "Person3", "Person4")
>
> colnames(adj_matrix) <- c("Event1", "Event2", "Event3", "Event4")
> print(adj_matrix)
        Event1 Event2 Event3 Event4
Person1      1      1      1      0
Person2      1      1      0      1
Person3      1      0      1      1
Person4      0      1      1      1
```

```
#Create a bipartite graph from the bladjacency matrix
bipartite_graph < graph_from_biadjacency_matrix(adj_matrix)
#Plot the bipartite graph
plot(bipartite_graph, main "Bipartite Network", vertex.size=30)
```


```
core_nodes < which (degree (bipartite_graph, mode "all") > 2)
periphery_nodes <- setdiff(V(bipartite_graph), core_nodes)
core_periphery - list(core core_nodes, periphery periphery_nodes)
print(core_periphery)
```

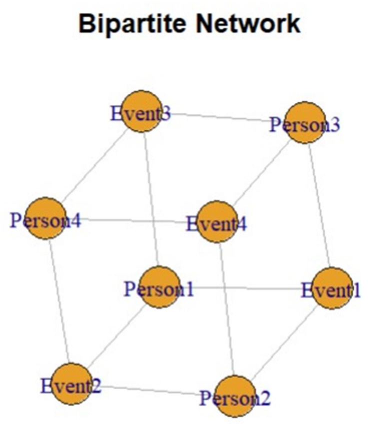
```
$core
Person1 Person2 Person3 Person4  Event1  Event2  Event3  Event4
      1       2       3       4       5       6       7       8

$periphery
integer(0)
```

Practical No:10

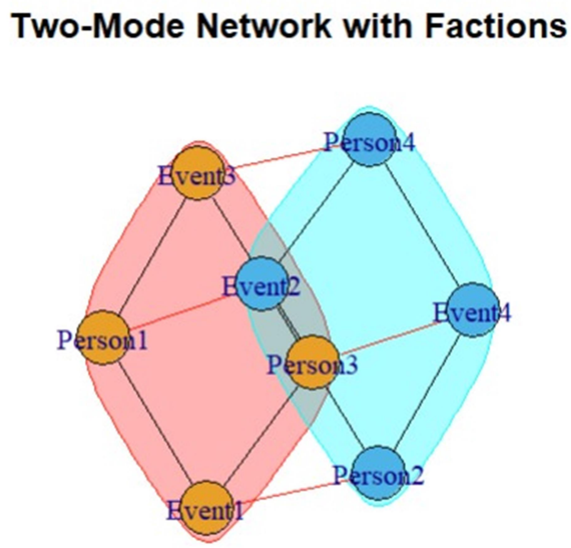
Aim: Find "factions" in the network using two-mode faction analysis.

Code:

```
library(bipartite)
library(igraph)

adj_matrix <- matrix(
  c(1, 1, 1, 0,
    1, 1, 0, 1,
    1, 0, 1, 1,
    0, 1, 1, 1),
  nrow = 4, ncol = 4, byrow = TRUE)

rownames(adj_matrix) <- c("Person1", "Person2", "Person3", "Person4")
colnames(adj_matrix) <- c("Event1", "Event2", "Event3", "Event4")

print(adj_matrix)

> print(adj_matrix)
        Event1 Event2 Event3 Event4
Person1      1      1      1      0
Person2      1      1      0      1
Person3      1      0      1      1
Person4      0      1      1      1

bipartite_graph <- graph_from_incidence_matrix(adj_matrix)
plot(bipartite_graph, main="Bipartite Network", vertex.size=30)
```




```
bipartite_community <- cluster_spinglass(bipartite_graph)

plot(bipartite_community, bipartite_graph,
    main = "Two-Mode Network with Factions",
    vertex.color = bipartite_community$membership + 1, vertex.size = 30)
```



factions <- bipartite_community$membership
print(factions)

```
[1] 2 1 2 1 2 1 2 1
```