

Social Network Analysis Journal

M.Sc Part I Computer Science

Ravi Yadav 542

March 22, 2023



R.J. College of Arts, Science & Commerce

Social network analysis

Seat number: 542



RAMNIRANJAN JHUNJHUNWALA COLLEGE
Ghatkopar (W), Mumbai-400 086

CERTIFICATE

M.Sc Part I
Computer Science
2022-2023

This is to certify that **Ravi Rameshchandra Yadav** from M.Sc Part I (Sem-II)
Computer Science, Seat No **542** satisfactorily completed the practicals of **SOCIAL NETWORK
ANALYSIS (PAPETRIII)** during the academic year **2022 - 2023** as specified by
the **MUMBAI UNIVERSITY**.

No. of Experiments completed **8** out of **8**

Sign of Incharge:

Date: March 22, 2023

Seat Number: **542**

Sign of Examiner:

Date: March 22, 2023

Course Co-ordinator

Contents

1	Practical 01	1
1.1	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.	1
1.1.1	1)no of edges	2
1.1.2	2)no of nodes	2
1.1.3	3)Degree Of nodes	3
1.1.4	4) a) Node with lowest degree	4
1.1.5	4) b) Node with lowest degree	4
1.1.6	5) To find neighbours / adjacency list:	4
1.1.7	6)Adjacency Matrix	4
2	Practical 02	5
2.1	Aim:	5
3	Practical 03	7
3.1	Aim:	7
3.1.1	1)Density	8
3.1.2	2) Degree	8
3.1.3	3)Reciprocity	8
3.1.4	Formula	9
3.1.5	4)Transitivity	10
3.1.6	5)Centralization	11
3.2	6) Clustering	13
4	Practical 04	14
4.1	Aim:	14
4.1.1	(i) Length of the shortest path from a given node to another node;	14
4.1.2	(ii) the density of the graph	16
5	Practical 05	17
5.1	Aim:	17
5.1.1	(i) Length of the shortest path from a given node to another node;	18
6	Practical 05	20
6.1	Aim:	20
6.1.1	(i) i) structural equivalence	21
6.2	ii) automorphic equivalence,	22
6.3	3) regular equivalence from a network.	23
7	Practical 07	24
7.1	Aim:	24
8	Practical 08	26
8.1	Aim:	26

1 Practical 01

- 1.1 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.

```
[1]: library(igraph)
```

Attaching package: ‘igraph’

The following objects are masked from ‘package:stats’:

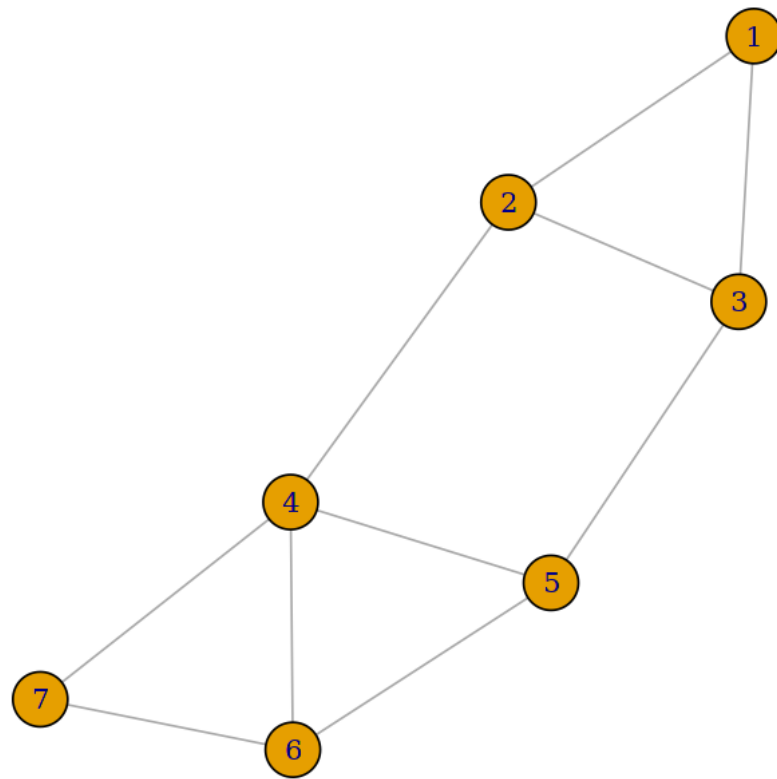
decompose, spectrum

The following object is masked from ‘package:base’:

union

```
[2]: g <- graph.formula(1-2, 1-3, 2-3, 2-4, 3-5, 4-5, 4-6,4-7, 5-6, 6-7)
```

```
[3]: plot(g)
```



1.1.1 1)no of edges

[4] : `ecount(g)`

10

1.1.2 2)no of nodes

[5] : `vcount(g)`

7

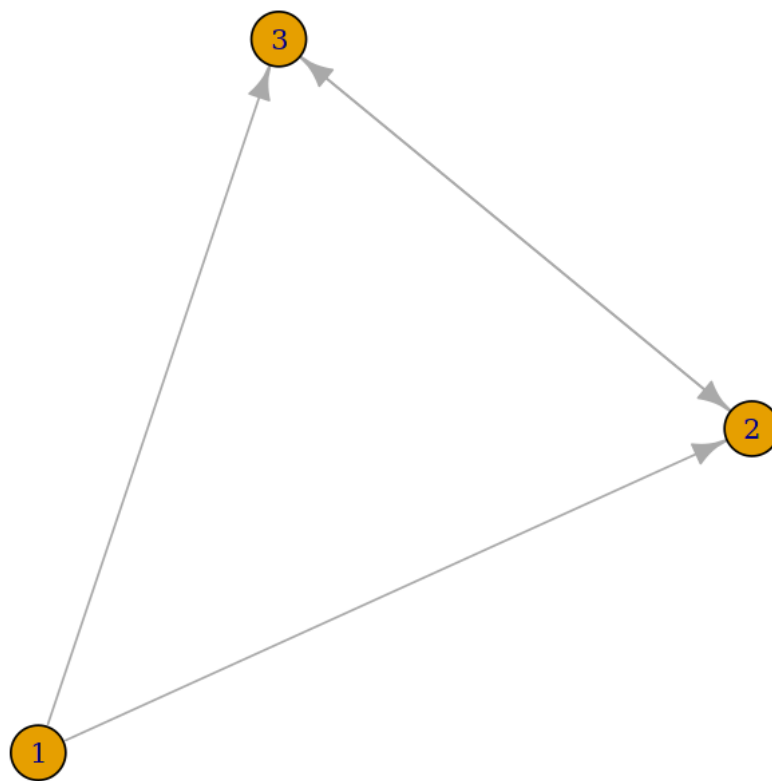
1.1.3 3)Degree Of nodes

```
[6]: degree(g)
```

```
1      2 2      3 3      3 4      4 5      3 6      3 7      2
```

```
[7]: dg <- graph.formula(1-+2, 1-+3, 2++3)
```

```
[8]: plot(dg)
```



```
[9]: degree(dg, mode="in")
```

```
1      0 2      2 3      2
```

```
[10]: degree(dg, mode="out")
```

```
1          2 2          1 3          1
```

1.1.4 4) a) Node with lowest degree

```
[11]: V(dg)$name[degree(dg)==min(degree(dg))]
```

```
'1'
```

1.1.5 4) b) Node with lowest degree

```
[12]: V(dg)$name[degree(dg)==max(degree(dg))]
```

```
1. '2' 2. '3'
```

1.1.6 5) To find neighbours / adjacency list:

```
[13]: neighbors(g,5)
```

```
+ 3/7 vertices, named, from 881bcb9:  
[1] 3 4 6
```

```
[14]: get.adjlist(dg)
```

```
$`1`  
+ 2/3 vertices, named, from d90acba:  
[1] 2 3
```

```
$`2`  
+ 3/3 vertices, named, from d90acba:  
[1] 1 3 3
```

```
$`3`  
+ 3/3 vertices, named, from d90acba:  
[1] 1 2 2
```

1.1.7 6)Adjacency Matrix

```
[15]: get.adjacency(g)
```

```
7 x 7 sparse Matrix of class "dgCMatrix"  
 1 2 3 4 5 6 7  
1 . 1 1 . . . .  
2 1 . 1 1 . . .  
3 1 1 . . 1 . .  
4 . 1 . . 1 1 1  
5 . . 1 1 . 1 .
```

```
6 . . . 1 1 . 1
7 . . . 1 . 1 .
```

2 Practical 02

2.1 Aim:

Perform following tasks:

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

```
[1]: library(igraph)
```

Attaching package: ‘igraph’

The following objects are masked from ‘package:stats’:

decompose, spectrum

The following object is masked from ‘package:base’:

union

```
[2]: nodes <- read.csv("/kaggle/input/network-analysis-data-from-various-sources/
↳ InputFileNodes.csv", header=T, , as.is=T)
```

```
[3]: head(nodes)
```

A data.frame: 6 × 5

	id	media	media.type	type.label	audience.size
	<chr>	<chr>	<int>	<chr>	<int>
1	s01	NY Times	1	Newspaper	20
2	s02	Washington Post	1	Newspaper	25
3	s03	Wall Street Journal	1	Newspaper	30
4	s04	USA Today	1	Newspaper	32
5	s05	LA Times	1	Newspaper	20
6	s06	New York Post	1	Newspaper	50

```
[4]: links <- read.csv("/kaggle/input/network-analysis-data-from-various-sources/
↳ InputFileEdges.csv", header=T, as.is=T)
```

```
[5]: head(links)
```


		from <chr>	to <chr>	weight <int>	type <chr>
A data.frame: 6 × 4	1	s01	s02	10	hyperlink
	2	s01	s02	12	hyperlink
	3	s01	s03	22	hyperlink
	4	s01	s04	21	hyperlink
	5	s04	s11	22	mention
	6	s05	s15	21	mention

```
[6]: net <- graph.data.frame(d=links, vertices=nodes, directed=T)
```

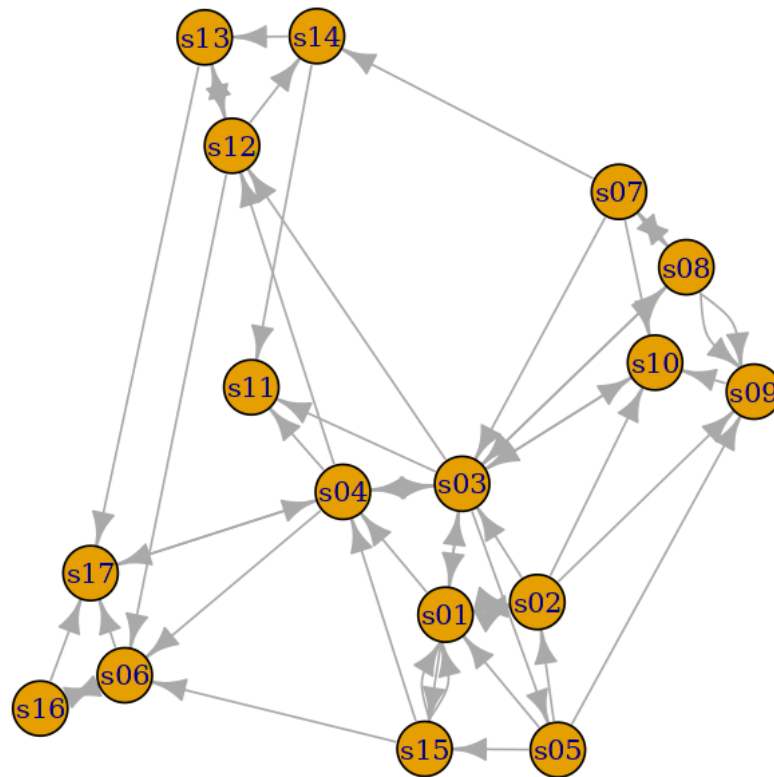
```
[7]: net <- graph.data.frame(d=links, vertices=nodes, directed=T)
m=as.matrix(net)

g <- graph.adjacency(m, mode="directed")

# Get adjacency matrix
A <- as.matrix(get.adjacency(g))
A
```

		s01	s02	s03	s04	s05	s06	s07	s08	s09	s10	s11	s12	s13	s14
A matrix: 17 × 17 of type dbl	s01	0	2	1	1	0	0	0	0	0	0	0	0	0	0
	s02	1	0	1	0	0	0	0	0	1	1	0	0	0	0
	s03	1	0	0	1	1	0	0	1	0	1	1	1	0	0
	s04	0	0	1	0	0	1	0	0	0	0	1	1	0	0
	s05	1	1	0	0	0	0	0	0	1	0	0	0	0	0
	s06	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	s07	0	0	1	0	0	0	0	1	0	1	0	0	0	1
	s08	0	0	1	0	0	0	1	0	2	0	0	0	0	0
	s09	0	0	0	0	0	0	0	0	0	1	0	0	0	0
	s10	0	0	1	0	0	0	0	0	0	0	0	0	0	0
	s11	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	s12	0	0	0	0	0	1	0	0	0	0	0	0	1	1
	s13	0	0	0	0	0	0	0	0	0	0	0	1	0	0
	s14	0	0	0	0	0	0	0	0	0	0	1	0	1	0
	s15	2	0	0	1	0	1	0	0	0	0	0	0	0	0
	s16	0	0	0	0	0	1	0	0	0	0	0	0	0	0
	s17	0	0	0	1	0	0	0	0	0	0	0	0	0	0

```
[8]: plot(net)
```



```
[ ]:
```

3 Practical 03

3.1 Aim:

Compute the following node level measures: (i) Density; (ii) Degree; (iii) Reciprocity; (iv) Tra

```
[1]: library(igraph)
```

Attaching package: 'igraph'

The following objects are masked from ‘package:stats’:

decompose, spectrum

The following object is masked from ‘package:base’:

union

3.1.1 1)Density

```
[2]: g <- graph.formula(1-2, 1-3, 2-3, 2-4, 3-5, 4-5, 4-6,4-7, 5-6, 6-7)
```

```
[3]: nodes <- read.csv("/kaggle/input/network-analysis-data-from-various-sources/
↳InputFileNodes.csv", header=T, , as.is=T)
links <- read.csv("/kaggle/input/network-analysis-data-from-various-sources/
↳InputFileEdges.csv", header=T, as.is=T)
net <- graph.data.frame(d=links, vertices=nodes, directed=T)
```

```
[4]: vcount(g)
```

7

```
[5]: ecount(g)
```

10

```
[6]: ecount(g)/(vcount(g)*(vcount(g)-1)/2)
```

0.476190476190476

3.1.2 2) Degree

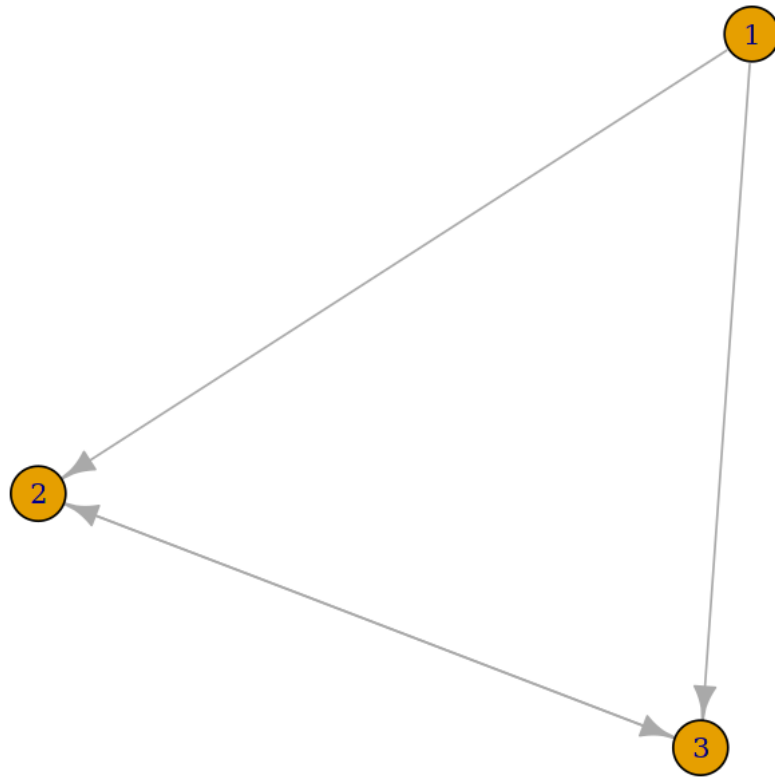
```
[7]: degree(net)
```

s01 10 s02 7 s03 13 s04 9 s05 5 s06 6 s07 5 s08 6 s09 5 s10 5 s11 3 s12 6 s13 4 s14 4 s15 6
s16 3 s17 5

3.1.3 3)Reciprocity

```
[8]: dg <- graph.formula(1-+2, 1-+3, 2++3)
plot(dg)
reciprocity(dg)
```

0.5



3.1.4 Formula

```
[9]: dyad.census(dg)
```

```
$mut 1
```

```
$asym 2
```

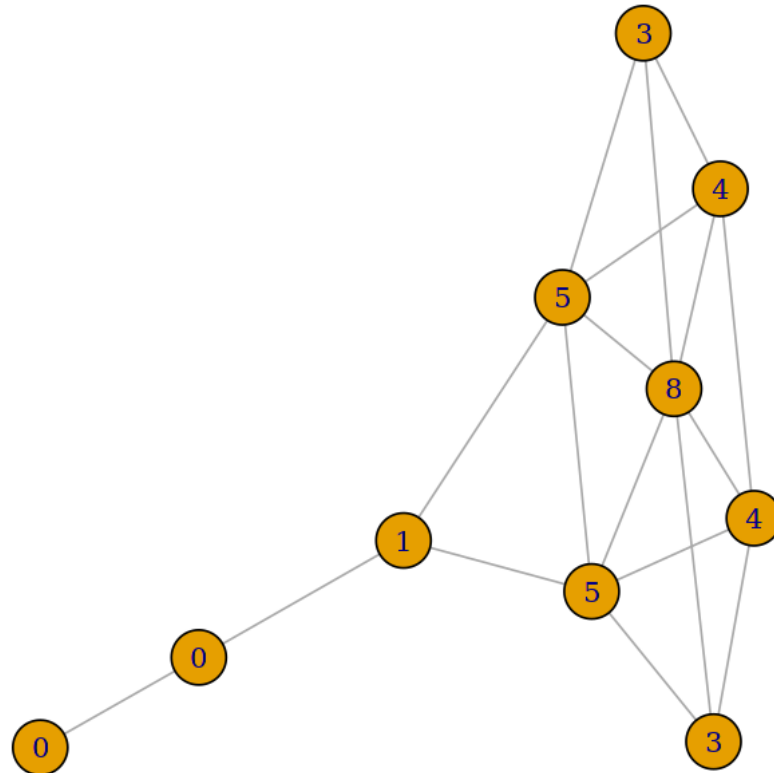
```
$null 0
```

```
[10]: 2*dyad.census(dg)$mut/ecount(dg)
```

```
0.5
```

3.1.5 4)Transitivity

```
[11]: kite <- graph.famous("Krackhardt_Kite")  
      atri <- adjacent.triangles(kite)  
      plot(kite, vertex.label=atri)
```



```
[12]: transitivity(kite, type="local")
```

```
1. 0.6666666666666667 2. 0.6666666666666667 3. 1 4. 0.5333333333333333 5. 1 6. 0.5 7. 0.5  
8. 0.3333333333333333 9. 0 10. NaN
```

Formula

```
[13]: adjacent.triangles(kite) / (degree(kite) * (degree(kite)-1)/2)
```

1. 0.6666666666666667 2. 0.6666666666666667 3. 1 4. 0.5333333333333333 5. 1 6. 0.5 7. 0.5
8. 0.3333333333333333 9. 0 10. NaN

3.1.6 5)Centralization

Degree of centrality

```
[14]: centralization.degree(net, mode="in", normalized=T)
```

\$res 1. 5 2. 3 3. 6 4. 4 5. 1 6. 4 7. 1 8. 2 9. 4 10. 4 11. 3 12. 3 13. 2 14. 2 15. 2 16. 1 17. 4

\$centralization 0.1875

\$theoretical_max 272

Closeness Centralization

```
[15]: closeness(net, mode="all", weights=NA)
centralization.closeness(net, mode="all", normalized=T)
```

s01 0.0333333333333333 s02 0.0303030303030303 s03 0.0416666666666667 s04
0.0384615384615385 s05 0.032258064516129 s06 0.03125 s07 0.0303030303030303 s08
0.0285714285714286 s09 0.0256410256410256 s10 0.0294117647058824 s11 0.032258064516129
s12 0.0357142857142857 s13 0.027027027027027 s14 0.0294117647058824 s15
0.0303030303030303 s16 0.0222222222222222 s17 0.0285714285714286

\$res 1. 0.5333333333333333 2. 0.484848484848485 3. 0.6666666666666667 4. 0.615384615384615
5. 0.516129032258065 6. 0.5 7. 0.484848484848485 8. 0.457142857142857 9. 0.41025641025641
10. 0.470588235294118 11. 0.516129032258065 12. 0.571428571428571 13. 0.432432432432432
14. 0.470588235294118 15. 0.484848484848485 16. 0.3555555555555556 17. 0.457142857142857

\$centralization 0.375359630727278

\$theoretical_max 7.74193548387097

Betweenness Centrality

```
[16]: betweenness(net, directed=T, weights=NA)
edge.betweenness(net, directed=T, weights=NA)
centralization.betweenness(net, directed=T, normalized=T)
```

s01 26.8571428571429 s02 6.23809523809524 s03 126.511904761905 s04 92.6428571428571 s05 13
s06 20.3333333333333 s07 1.75 s08 21 s09 1 s10 15 s11 0 s12 33.5 s13 20 s14 4 s15
5.66666666666667 s16 0 s17 58.5

1. 6.61904761904762 2. 6.61904761904762 3. 11.7857142857143 4. 8.33333333333333 5. 6.5
6. 11.6666666666667 7. 21.3333333333333 8. 4.25 9. 4.25 10. 16 11. 64.4761904761905 12. 9.5
13. 3.26190476190476 14. 3.26190476190476 15. 15 16. 1 17. 15 18. 17 19. 16.75 20. 2 21. 1.25
22. 8 23. 12.5 24. 4 25. 26 26. 18 27. 14.5 28. 17 29. 7.5 30. 4.5 31. 2.73809523809524 32. 23 33. 11
34. 31 35. 9.01190476190476 36. 18 37. 28.5 38. 3 39. 6.5 40. 17 41. 8.66666666666667 42. 74.5
43. 11.75 44. 34 45. 4.5 46. 6.33333333333333 47. 8.80952380952381 48. 5.33333333333333 49. 3
50. 28 51. 10

```
$res 1. 26.8571428571429 2. 6.23809523809524 3. 126.511904761905 4. 92.6428571428571 5. 13
6. 20.3333333333333 7. 1.75 8. 21 9. 1 10. 15 11. 0 12. 33.5 13. 20 14. 4 15. 5.66666666666667
16. 0 17. 58.5
```

```
$centralization 0.443932911706349
```

```
$theoretical_max 3840
```

```
Eigenvector centrality
```

```
[17]: centralization.evcent(net, directed=T, normalized=T)
```

```
$vector 1. 0.777185829200523 2. 0.569523129226997 3. 1 4. 0.821414404772152
5. 0.306115118060718 6. 0.605185074708371 7. 0.103395270890436 8. 0.337765973616263
9. 0.47483664722783 10. 0.657460289883597 11. 0.627101587234399 12. 0.638699752169925
13. 0.265054751720928 14. 0.227166505596393 15. 0.331614797366162 16. 0.185256300592937
17. 0.574550689029643
```

```
$value 3.26674489758997
```

```
$options $bmat 'I'
```

```
$n 17
```

```
$which 'LR'
```

```
$nev 1
```

```
$tol 0
```

```
$ncv 0
```

```
$ldv 0
```

```
$ishift 1
```

```
$maxiter 3000
```

```
$nb 1
```

```
$mode 1
```

```
$start 1
```

```
$sigma 0
```

```
$sigmai 0
```

```
$info 0
```

```
$iter 7
```

```
$nconv 1
```

```
$numop 30
```

```
$numopb 0
```

```
$numreo 20
```

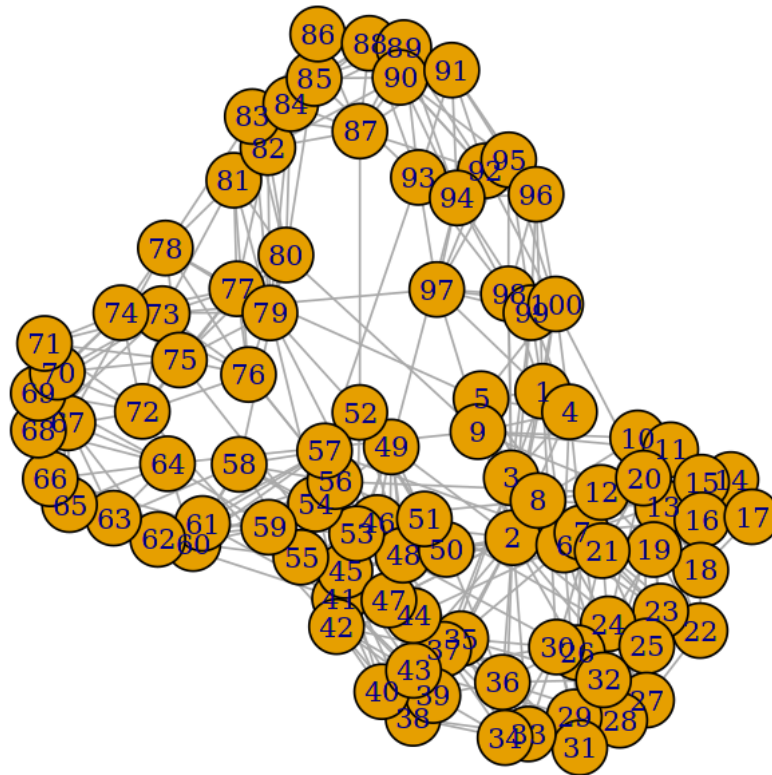
```
$centralization 0.53110461741892
```

\$theoretical_max 16

3.2 6) Clustering

```
[18]: # let's generate two networks and merge them into one graph.
g2 <- barabasi.game(50, p=2, directed=F)
g1 <- watts.strogatz.game(1, size=100, nei=5, p=0.05)
g <- graph.union(g1,g2)

#Let's remove multi-edges and loops
g <- simplify(g)
plot(g)
```



4 Practical 04

4.1 Aim:

For a given network find the following: (i) Length of the shortest path from a given node to another node;

```
[1]: library(igraph)
```

Attaching package: 'igraph'

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

decompose, spectrum

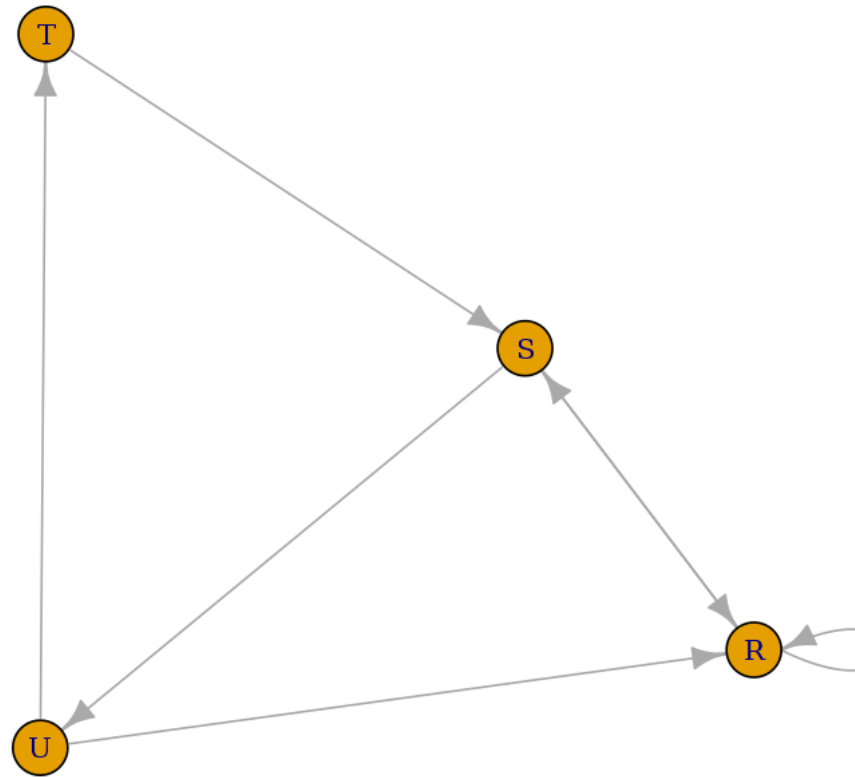
The following object is masked from 'package:base':

union

4.1.1 (i) Length of the shortest path from a given node to another node;

```
[2]: matt <- as.matrix(read.table(text=
  "node R S T U
      R 7 5 0 0
      S 7 0 0 2
      T 0 6 0 0
      U 4 0 1 0", header=T))
```

```
[3]: nms <- matt[,1 ]
matt <- matt[, -1]
colnames(matt) <- rownames(matt) <- nms
matt[is.na(matt)] <- 0
g <- graph.adjacency(matt, weighted=TRUE)
plot(g)
```



```
[4]: s.paths <- shortest.paths(g, algorithm = "dijkstra")
      print(s.paths)
```

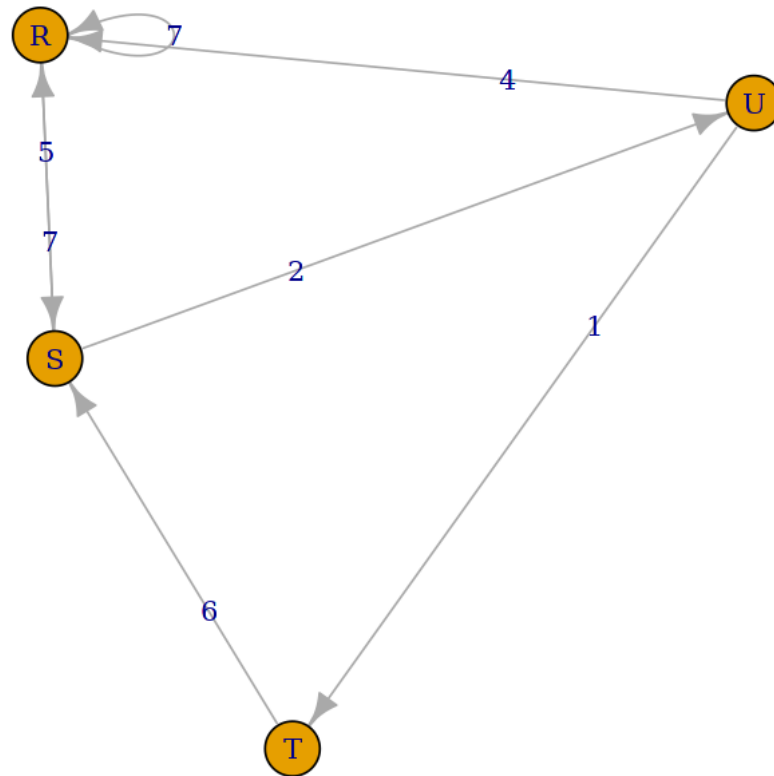
```

      R S T U
R 0 5 5 4
S 5 0 3 2
T 5 3 0 1
U 4 2 1 0
```

```
[5]: shortest.paths(g, v="R", to="S")
```

A matrix: 1×1 of type dbl $\frac{R}{S} \mid \frac{5}{5}$

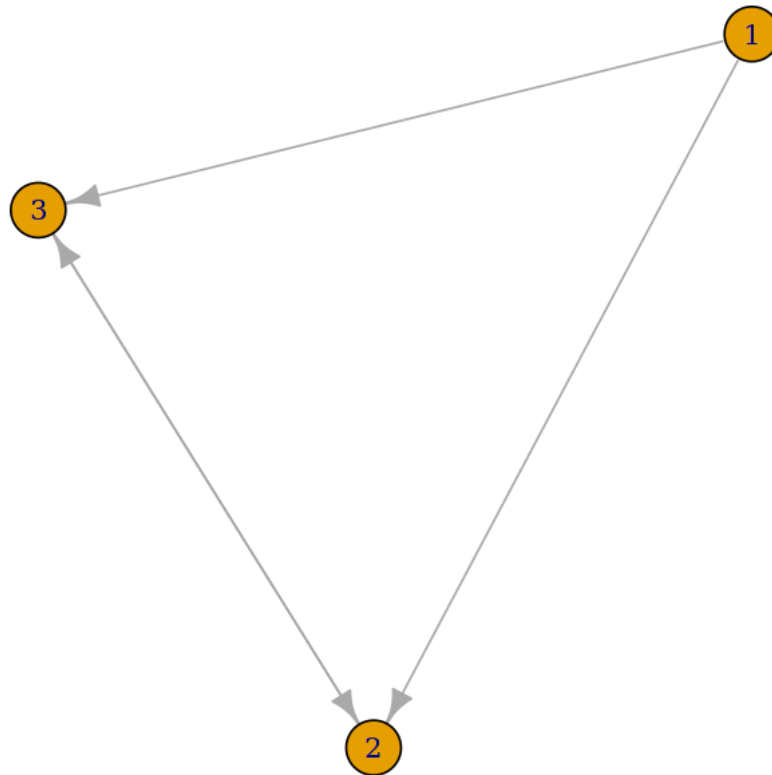
```
[6]: plot(g, edge.label=E(g)$weight)
```



4.1.2 (ii) the density of the graph

```
[7]: dg <- graph.formula(1-+2, 1-+3, 2++3)
plot(dg)
graph.density(dg, loops=TRUE)
```

0.4444444444444444



```
[8]: graph.density(simplify(dg), loops=FALSE)
```

```
0.6666666666666667
```

5 Practical 05

5.1 Aim:

Write a program to distinguish between a network as a matrix, a network as an edge list, and a network as a sociogram (or ‘network graph’)

```
[1]: library(igraph)
```

Attaching package: ‘igraph’

The following objects are masked from ‘package:stats’:

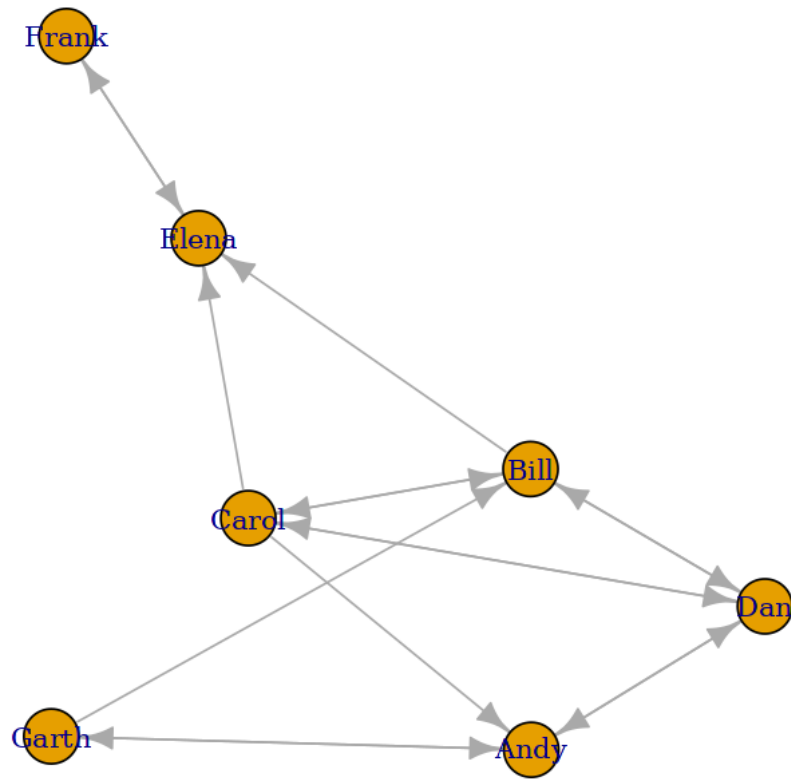
decompose, spectrum

The following object is masked from ‘package:base’:

union

5.1.1 (i) Length of the shortest path from a given node to another node;

```
[2]: ng<-graph.  
      ↪formula(Andy++Garth,Garth-+Bill,Bill-+Elena,Elena++Frank,Carol-+Andy,Carol-+Elena,Carol++Dan,  
[3]: plot(ng)
```



2) a network as a matrix,

```
[4]: get.adjacency(ng)
```

```

7 x 7 sparse Matrix of class "dgCMatrix"
      Andy Garth Bill Elena Frank Carol Dan
Andy    .     1    .     .     .     .     1
Garth    1     .     1     .     .     .     .
Bill     .     .     .     1     .     1     1
Elena    .     .     .     .     1     .     .
Frank    .     .     .     1     .     .     .
Carol    1     .     1     1     .     .     1
Dan      1     .     1     .     .     1     .

```

iii) a network as an edge list.

```
[5]: E(ng)
```

```
+ 16/16 edges from 9022c9b (vertex names):  
[1] Andy ->Garth Andy ->Dan Garth->Andy Garth->Bill Bill ->Elena  
[6] Bill ->Carol Bill ->Dan Elena->Frank Frank->Elena Carol->Andy  
[11] Carol->Bill Carol->Elena Carol->Dan Dan ->Andy Dan ->Bill  
[16] Dan ->Carol
```

```
[6]: get.adjedgelist(ng,mode="in")
```

```
$Andy  
+ 3/16 edges from 9022c9b (vertex names):  
[1] Garth->Andy Carol->Andy Dan ->Andy  
  
$Garth  
+ 1/16 edge from 9022c9b (vertex names):  
[1] Andy->Garth  
  
$Bill  
+ 3/16 edges from 9022c9b (vertex names):  
[1] Garth->Bill Carol->Bill Dan ->Bill  
  
$Elena  
+ 3/16 edges from 9022c9b (vertex names):  
[1] Bill ->Elena Frank->Elena Carol->Elena  
  
$Frank  
+ 1/16 edge from 9022c9b (vertex names):  
[1] Elena->Frank  
  
$Carol  
+ 2/16 edges from 9022c9b (vertex names):  
[1] Bill->Carol Dan ->Carol  
  
$Dan  
+ 3/16 edges from 9022c9b (vertex names):  
[1] Andy ->Dan Bill ->Dan Carol->Dan
```

6 Practical 05

6.1 Aim:

Write a program to distinguish between a network as a matrix, a network as an edge list, and a network as a sociogram (or “network graph”)

```
[1]: install.packages("sna")  
install.packages("network")
```

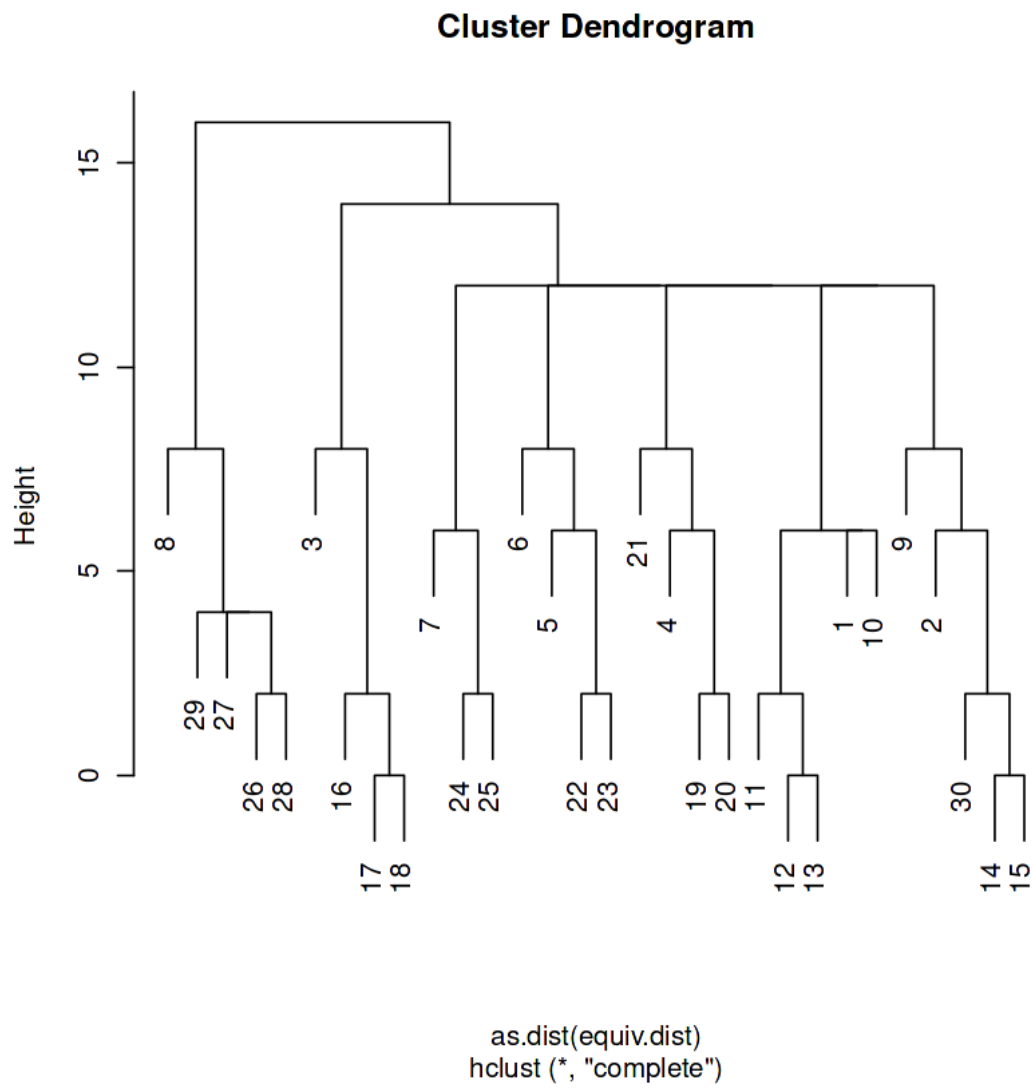
Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)

Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)

```
[4]: library(sna)  
library(igraph)
```

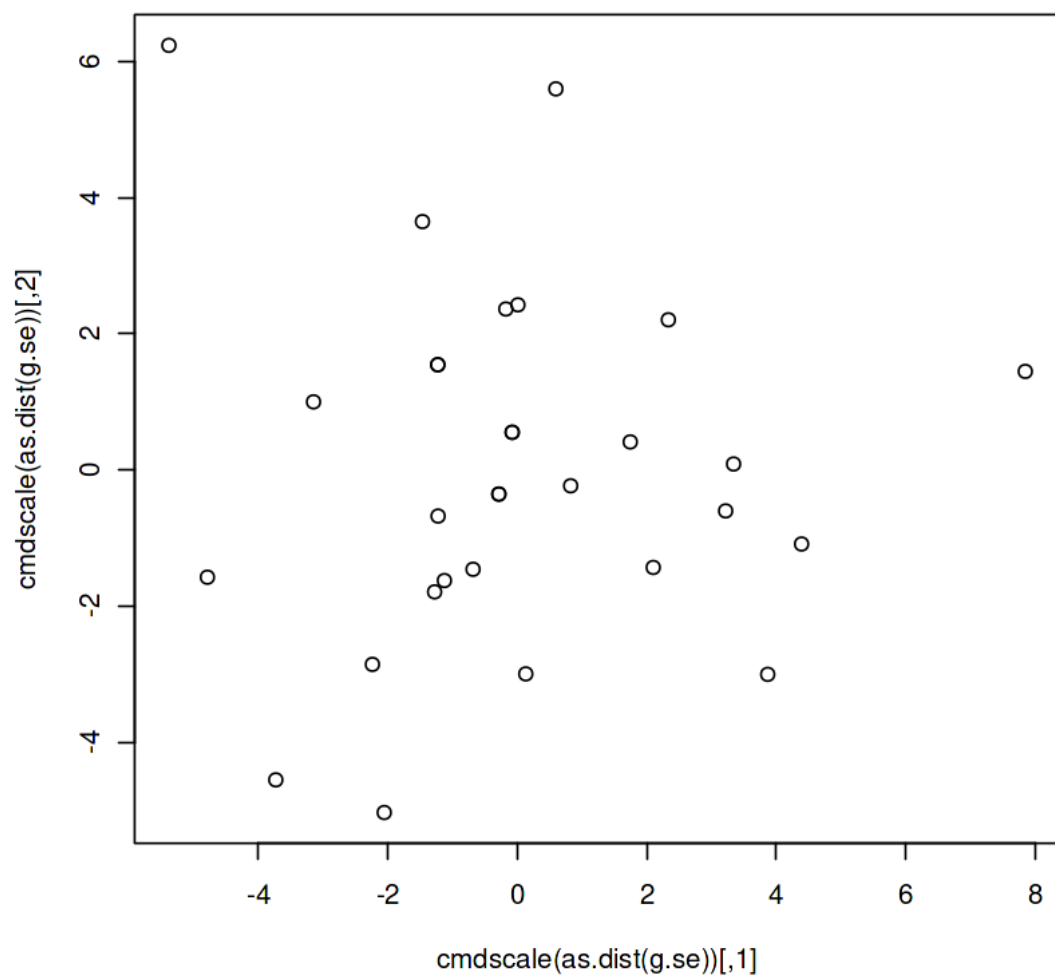
6.1.1 (i) i) structural equivalence

```
[6]: links2 <- read.csv("/kaggle/input/sna-edges/edges1.csv", header=T, row.names=1)  
eq<-equiv.clust(links2)  
plot(eq)
```

6.2 ii) automorphic equivalence,

```
[7]: g.se<-sedist(links2)
      #Plot a metric MDS of vertex positions in two dimensions
      plot(cmdscale(as.dist(g.se)))
```

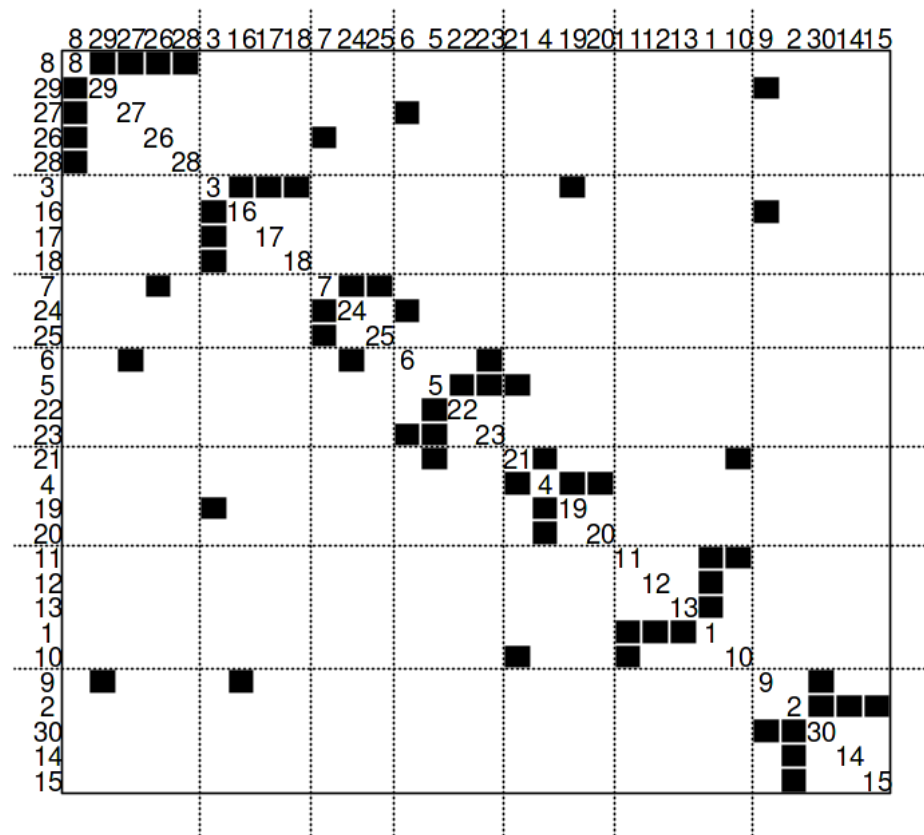


6.3 3) regular equivalence from a network.

Blockmodeling

```
[8]: b<-blockmodel(links2,eq,h=10)
      plot(b)
```

Relation - 1



7 Practical 07

7.1 Aim:

Create sociograms for the persons-by-persons network and the committee-by-committee network for a

```
[1]: library(igraph)
```

Attaching package: 'igraph'

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

```
decompose, spectrum
```

The following object is masked from ‘package:base’:

```
union
```

```
[2]: # Create sample data for data_Network_1
data_Network_1 <- data.frame(
  Source = c(1, 1, 2, 2, 2, 2, 2, 3, 3, 3),
  Target = c(2, 3, 1, 3, 4, 5, 6, 2, 4, 5)
)

# Create graph object
g <- graph_from_data_frame(data_Network_1, directed = TRUE)
```

```
[3]: # Set binary code for edges to be displayed
bytes <- "0011111111000000000"

# Extract edges based on binary code
edges <- which(strsplit(bytes, "")[[1]] == "1")

# Get layout for visualization
layout <- layout_with_kk(g)
```

```
[4]: library(dplyr)

# Plot sociogram
plot(g, layout = layout, edge.color = if_else(E(g)$id %in% edges, "red", "gray"))
```

Attaching package: ‘dplyr’

The following objects are masked from ‘package:igraph’:

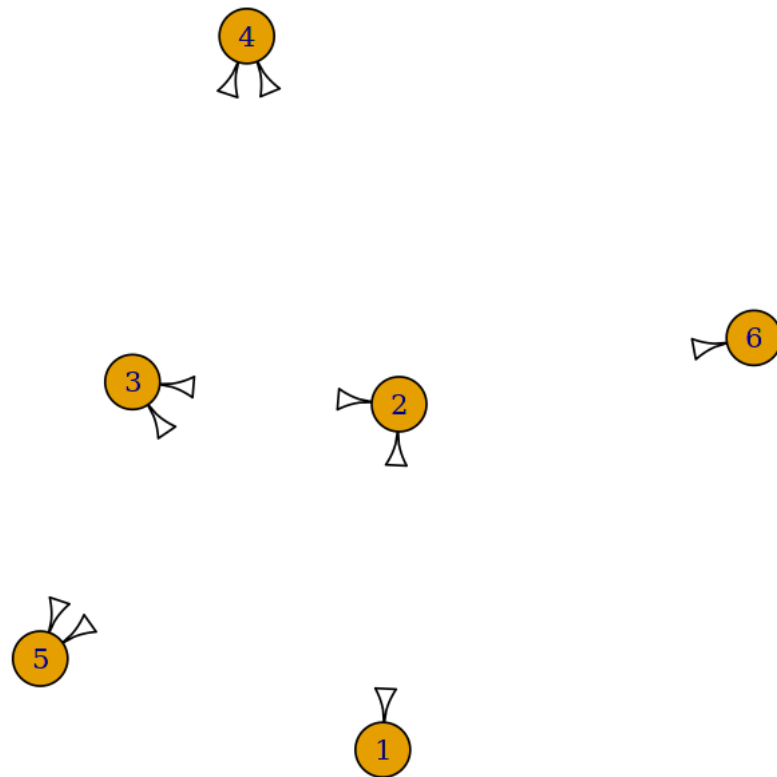
```
as_data_frame, groups, union
```

The following objects are masked from ‘package:stats’:

```
filter, lag
```

The following objects are masked from ‘package:base’:

`intersect, setdiff, setequal, union`



[]:

8 Practical 08

8.1 Aim:

Perform SVD analysis of a network.

```
[1]: library(igraph)
```

Attaching package: 'igraph'

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

decompose, spectrum

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

union

```
[2]: a <- matrix(c(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0,
                  0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1), 9, 4)
```

```
[3]: print(a)
```

	[,1]	[,2]	[,3]	[,4]
[1,]	1	1	0	0
[2,]	1	1	0	0
[3,]	1	1	0	0
[4,]	1	0	1	0
[5,]	1	0	1	0
[6,]	1	0	1	0
[7,]	1	0	0	1
[8,]	1	0	0	1
[9,]	1	0	0	1

```
[4]: svd(a)
```

\$d 1. 3.46410161513775 2. 1.73205080756888 3. 1.73205080756888 4. 1.35973995551052e-16

-0.3333333	0.4714045	-3.202997e-16	3.693981e-01
-0.3333333	0.4714045	-3.415421e-16	4.459029e-01
-0.3333333	0.4714045	8.520300e-18	-8.153010e-01
-0.3333333	-0.2357023	-4.082483e-01	7.849070e-17
-0.3333333	-0.2357023	-4.082483e-01	1.340019e-16
-0.3333333	-0.2357023	-4.082483e-01	1.340019e-16
-0.3333333	-0.2357023	4.082483e-01	1.182076e-16
-0.3333333	-0.2357023	4.082483e-01	1.182076e-16
-0.3333333	-0.2357023	4.082483e-01	1.182076e-16

\$u A matrix: 9 × 4 of type dbl

```

-0.8660254  0.0000000  -4.378026e-17  0.5
$V A matrix: 4 x 4 of type dbl -0.2886751  0.8164966  -2.509507e-16  -0.5
-0.2886751  -0.4082483  -7.071068e-01  -0.5
-0.2886751  -0.4082483  7.071068e-01  -0.5
```

```
[ ]:
```