# Basis SNA in R

Robin Khalfa

2025-05-21

## 1. Inleiding

Sociale netwerkanalyse (SNA) is een analytische benadering die sociale structuren beschouwt in termen van nodes (actoren) en edges (relaties). In tegenstelling tot traditionele benaderingen binnen de sociale wetenschappen, die voornamelijk focussen op de kenmerken van individuele eenheden, staat binnen SNA *relationaliteit* centraal: de betekenisvolle patronen van interactie en verbinding tussen actoren.

Netwerken kunnen verschillende vormen aannemen, van vriendschapsrelaties en samenwerkingsverbanden tot informatiestromen en institutionele connecties, en zijn fundamenteel voor het begrijpen van dynamieken zoals macht, invloed, cohesie en fragmentatie.

Met behulp van de programmeertaal **R** kan men op efficiënte wijze netwerken construeren, visualiseren en analyseren alsook analyses automatiseren. In deze module leggen we de nadruk op **beschrijvende netwerkstatistieken**, gestructureerd volgens vijf klassieke niveaus van analyse in SNA:

1. **Node level** – individuele actoren en hun structurele kenmerken
2. **Dyad level** – relaties tussen twee actoren
3. **Triad level** – driehoeksstructuren en hun implicaties
4. **Tussen-niveau** – substructuren, rollen en posities
5. **Netwerkniveau** – globale netwerkkenmerken

Deze indeling biedt een systematisch raamwerk om de complexiteit van sociale netwerken te ontleden. Elk niveau biedt unieke inzichten en draagt bij aan een begrip van netwerken als sociale systemen.

De volgende secties demonstreren we de computationele implementatie van deze analyses in R. We werken doorheen deze notebook met de `class555_matrix.csv` dataset die we in de vorige notebook reeds hebben behandeld.

In een laatste bonus deel tonen we ook hoe statistische netwerkenmodellen kunnen worden gedraaid in R. Dit vormt niet de kern van deze notebook, maar kan gezien worden als een extra bonus deel voor de geïntresseerden onder ons :).

## 2. Definiëren van het netwerk

Aan de start van een analyse is het altijd goed om even zicht te krijgen op een aantal zaken en het netwerk te beschrijven. Hieronder een overzicht van een aantal belangrijke beschrijvende vragen/eigenschappen:

- Nog even inladen indien nodig en omzetten naar igraph:

```
# Inlezen van de data
#library(igraph)
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 4.3.3
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
class_mat <- read.csv(file = "Data/class555_matrix.csv", header = TRUE)
class_mat <- as.matrix(class_mat)
rownames(class_mat) <- 1:nrow(class_mat)
colnames(class_mat) <- 1:ncol(class_mat)

class_matrix_net <- igraph::graph_from_adjacency_matrix(adjmatrix = class_mat,
                                                        mode = "directed")

class_att <- read.csv(file = "Data/class555_attributedata.csv", header = TRUE)

class_matrix_net <- igraph::set_vertex_attr(graph = class_matrix_net,
                                    name = "gender",
                                    value = class_att$gender)


class_matrix_net <- igraph::set_vertex_attr(graph = class_matrix_net,
                                    name = "grade",
                                    value = class_att$grade)

class_matrix_net <- igraph::set_vertex_attr(graph = class_matrix_net,
                                    name = "race",
                                    value = class_att$race)
class_matrix_net
```

```
## IGRAPH b0bf9b4 DN-- 24 77 --
## + attr: name (v/c), gender (v/c), grade (v/n), race (v/c)
## + edges from b0bf9b4 (vertex names):
##  [1] 1 ->3  1 ->5  1 ->7  1 ->21 2 ->3  2 ->6  3 ->6  3 ->8  3 ->16 3 ->24
## [11] 4 ->13 4 ->18 7 ->1  7 ->9  7 ->10 7 ->16 8 ->3  8 ->9  8 ->13 9 ->5
## [21] 9 ->8  10->6  10->14 10->19 10->20 10->24 11->12 11->15 11->18 11->24
## [31] 12->11 12->15 12->24 13->8  14->10 14->13 14->19 14->21 14->24 15->10
## [41] 15->11 15->13 15->14 15->24 16->3  16->5  16->9  16->19 17->8  17->13
## [51] 17->18 17->23 17->24 18->13 18->17 18->23 18->24 19->14 19->16 19->20
## [61] 19->21 20->19 20->21 20->24 21->5  21->19 21->20 22->23 23->5  23->13
## [71] 23->17 23->18 24->6  24->10 24->14 24->15 24->21
```

- Hoeveel nodes telt het netwerk? (= aantal rijen van de matrix):

```
#detach(package:sna)
print(paste("Het netwerk omvat", nrow(as.matrix(class_matrix_net)), "nodes"))
```

```
## [1] "Het netwerk omvat 24 nodes"
```

- Hoeveel edges en mogelijke edges telt het netwerk?:

```
#Aantal edges (base R - geen igraph object)
print(paste("Het netwerk omvat", sum(as.matrix(igraph::as_adjacency_matrix(class_matrix_net, sparse = T
```

```
## [1] "Het netwerk omvat 77 edges"
```

```
# OF Aantal edges (igraph objects)
print(paste("Het netwerk omvat", igraph::gsize(class_matrix_net), "edges"))
```

```
## [1] "Het netwerk omvat 77 edges"
```

```
# Of het totaal aantal mogelijke edges (nties = sna package)
print(paste("Het netwerk omvat", sna::nties(as.matrix(igraph::as_adjacency_matrix(class_matrix_net, spa
```

```
## [1] "Het netwerk omvat 552  mogelijke edges"
```

```
# Dit laatste is gelijk aan
(nrow(as.matrix(igraph::as_adjacency_matrix(class_matrix_net, sparse = TRUE))) * ncol(as.matrix(igraph:
```

```
## [1] 552
```

- Gaat het om een directed of undirected netwerk? (is directed?)

```
igraph::is_directed(class_matrix_net)
```

```
## [1] TRUE
```

- Welke attributen omvat het netwerk?

```
# Vertex (node) attributen
igraph::vertex_attr_names(class_matrix_net)
```

```
## [1] "name"   "gender" "grade"  "race"
```

```
# Edge attributen
igraph::edge_attr_names(class_matrix_net)
```

```
## character(0)
```

- Hoeveel wederkerige relaties telt het netwerk?

```r
sum(igraph::which_mutual(class_matrix_net))
```

```
## [1] 38
```

# 3. Node level

Op het node niveau leggen we de focus op individuele actoren (nodes) en hun structurele positie binnen het netwerk. Deze positie is bepalend voor toegang tot informatie, invloed, afhankelijkheid en andere sociaal relevante uitkomsten. Hieronder behandelen we de meest courante centrale maten op dit niveau:

## 3.1 Degree centrality

De **degree centrality** betreft het aantal directe connecties dat een actor heeft. In een directed netwerk maken we een onderscheid tussen:

- **In-degree**: het aantal inkomende verbindingen
- **Out-degree**: het aantal uitgaande verbindingen

Deze maat is een directe indicator van activiteit (out-degree) of populariteit (in-degree):

```r
#detach(package:sna)

# In- en out-degree
in_deg <- igraph::degree(class_matrix_net, mode = "in")   # in-degree
out_deg <- igraph::degree(class_matrix_net, mode = "out")  # out-degree

in_deg
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
##  1  0  4  0  5  4  1  4  3  4  2  1  7  4  3  3  2  4  5  3  5  0  3  9
```

```r
print("**********************************************************************")
```

```
## [1] "**********************************************************************"
```

```r
out_deg
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
##  4  2  4  2  0  0  4  3  2  5  4  3  1  5  5  4  5  4  4  3  3  1  4  5
```

We kunnen dit ook heel handig weergeven in een tabel, alsook de totale degree:

```r
#detach(package:sna)
in_deg  <- igraph::degree(class_matrix_net, mode = "in")
out_deg <- igraph::degree(class_matrix_net, mode = "out")
tot_deg <- igraph::degree(class_matrix_net) #total degree (sum of in en outdegree)
```

```r
# Combineer in één data.frame
degree_tbl <- data.frame(
  node = names(in_deg),
  indegree = in_deg,
  outdegree = out_deg,
  degree = tot_deg
)  %>% arrange(desc(degree))

# Bekijk de topnodes
print(head(degree_tbl))
```

```
##    node indegree outdegree degree
## 24   24        9         5     14
## 10   10        4         5      9
## 14   14        4         5      9
## 19   19        5         4      9
## 3     3        4         4      8
## 13   13        7         1      8
```

Handig is ook om gemiddelde degrees te berekenen:

```r
#detach(package:sna)
mean(igraph::degree(class_matrix_net, mode = "in"))
```

```
## [1] 3.208333
```

```r
mean(igraph::degree(class_matrix_net, mode = "out"))
```

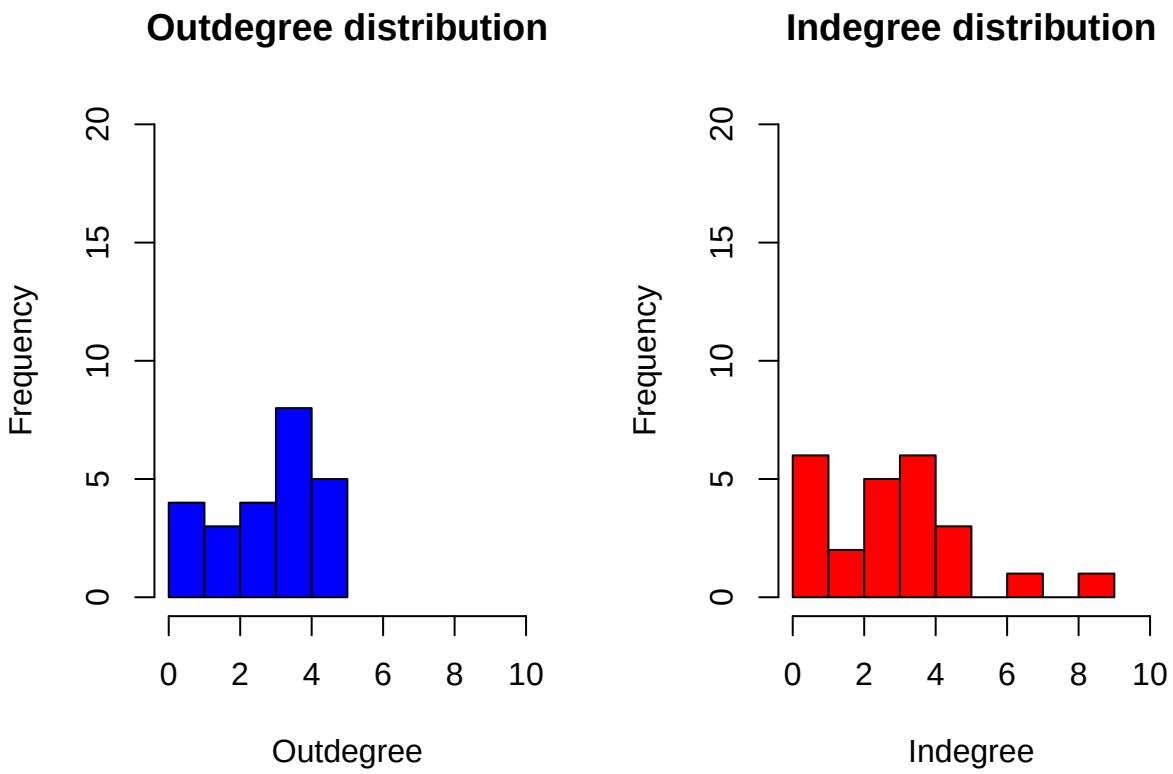
```
## [1] 3.208333
```

```r
mean(igraph::degree(class_matrix_net, mode = "all"))
```

```
## [1] 6.416667
```

Ook altijd handig om dit even in een histogram te gieten:

```r
#detach(package:sna)
par(mfrow=c(1,2))
hist(out_deg, xlim=c(0,10), ylim=c(0,20), breaks=7,
     main="Outdegree distribution", xlab="Outdegree", col="blue")
hist(in_deg, xlim=c(0,10), ylim=c(0,20), breaks=7,
     main="Indegree distribution", xlab="Indegree", col="red")
```




### 3.2 Betweenness Centrality

Betweenness centrality geeft aan hoe vaak een actor op het kortste pad tussen andere actoren ligt. Het meet de potentie van een actor om informatie of interacties te controleren of te bemiddelen:

```
#detach(package:sna)
bet_cent <- igraph::betweenness(class_matrix_net, directed = TRUE)
print(bet_cent)
```

```
##          1          2          3          4          5          6          7
##   1.833333   0.000000  84.683333   0.000000   0.000000   0.000000   2.833333
##          8          9         10         11         12         13         14
##  60.366667   5.533333  25.450000  68.000000   0.000000  23.766667  20.950000
##         15         16         17         18         19         20         21
##  91.700000  23.883333  16.700000  60.000000  28.550000  12.700000  18.533333
##         22         23         24
##   0.000000  21.500000 162.016667
```

```
# Calculate and print mean betweenness centrality - network level properties
mean_between <- mean(igraph::betweenness(class_matrix_net, directed = TRUE))
print(paste("De gemiddelde betweenness centrality is gelijk aan", mean_between))
```

```
## [1] "De gemiddelde betweenness centrality is gelijk aan 30.375"
```

## 3.3. Closeness Centrality

Closeness centrality meet hoe dicht een actor gemiddeld bij alle andere actoren in het netwerk staat, berekend als het omgekeerde van de som van de kortste padlengtes. Dit reflecteert de snelheid waarmee een actor informatie kan verspreiden of ontvangen.

```
#detach(package:sna)
# Closeness - in: Hoe snel actor info ontvangt:
close_cent_in <- igraph::closeness(class_matrix_net, mode = 'in', normalized = TRUE)
print(close_cent_in)
```

```
##         1         2         3         4         5         6         7         8
## 1.0000000       NaN 0.3888889       NaN 0.4782609 0.4583333 1.0000000 0.4375000
##         9        10        11        12        13        14        15        16
## 0.3750000 0.4200000 0.2876712 0.2258065 0.5121951 0.4285714 0.3888889 0.3684211
##        17        18        19        20        21        22        23        24
## 0.2234043 0.2763158 0.3888889 0.3442623 0.4468085       NaN 0.2258065 0.5833333
```

```
# Closeness - out: Hoe snel actor info kan verspreiden:
close_cent_out <- igraph::closeness(class_matrix_net, mode = 'out', normalized = TRUE)
print(close_cent_out)
```

```
##         1         2         3         4         5         6         7         8
## 0.3636364 0.3064516 0.4090909 0.3454545       NaN       NaN 0.3703704 0.3214286
##         9        10        11        12        13        14        15        16
## 0.2535211 0.3913043 0.4285714 0.4000000 0.2465753 0.4090909 0.4864865 0.3529412
##        17        18        19        20        21        22        23        24
## 0.4864865 0.4390244 0.3529412 0.3673469 0.3157895 0.2753623 0.3600000 0.4186047
```

```
# Calculate and print mean betweenness centrality - network level properties
mean_closeness_in <- mean(close_cent_in, na.rm = TRUE) # disregard missing values
print(paste("De gemiddelde afstand naar node vanaf andere nodes is gelijk aan", mean_closeness_in))
```

```
## [1] "De gemiddelde afstand naar node vanaf andere nodes is gelijk aan 0.440874132984209"
```

```
# Calculate and print mean betweenness centrality - network level properties
mean_closeness_out <- mean(close_cent_out, na.rm = TRUE) # disregard missing values
print(paste("De gemiddelde afstand van node naar andere nodes is gelijk aan", mean_closeness_out)) # st
```

```
## [1] "De gemiddelde afstand van node naar andere nodes is gelijk aan 0.368203573919085"
```

## 3.4. Eigenvector Centrality

Eigenvector centrality houdt niet enkel rekening met het aantal verbindingen, maar ook met de centraliteit van de knopen waarmee men verbonden is. Het meet dus invloed via invloedrijke anderen: Hoeveel centrale buren heeft een node?

```
#detach(package:sna)
# Eigenvector centrality
table(sort(igraph::eigen_centrality(class_matrix_net)$vector))
```

```
##
## 0.0369976204671164 0.0809241733638766  0.106643116179866  0.169469179884173
##                  1                  1                  1                  1
##  0.201059120861729  0.202906930440519  0.213088170979043   0.28005005040391
##                  1                  1                  1                  1
##  0.290307194568436  0.291660666758627  0.319308510884389  0.339354643785313
##                  1                  1                  1                  1
##  0.357893635222491  0.397942002654508  0.417945043444551   0.42274692478304
##                  1                  1                  1                  1
##  0.428816772038706  0.550393176171027  0.598017924696714  0.659548156000283
##                  1                  1                  1                  1
##  0.686979189315653  0.768880508098313  0.835600690963002                  1
##                  1                  1                  1                  1
```

```
# Gemiddelde:
mean_eig_cent <- mean(igraph::eigen_centrality(class_matrix_net)$vector)
print(paste("De gemiddelde eigenvectorcentraliteit van de nodes in het netwerk bedraagt", mean_eig_cent))
```

```
## [1] "De gemiddelde eigenvectorcentraliteit van de nodes in het netwerk bedraagt 0.40235555841522"
```

# 4. Dyad level

Binnen sociale netwerkanalyse (SNA) vormt het dyadische niveau (of dyad level) een fundamenteel analytisch perspectief. Een dyade verwijst naar het paargewijs verband tussen twee actoren (nodes) in een netwerk. In tegenstelling tot het individuele niveau (nodale kenmerken) of het globale netwerkniveau (structuureigenschappen zoals densiteit of centralisatie), focust de dyadische benadering op de interactie of relatie tussen twee specifieke entiteiten, ongeacht de bredere netwerkcontext.

Dyades vormen de bouwstenen van netwerken. Elke dyade kan worden gekarakteriseerd door de aanwezigheid of afwezigheid van een relatie (binaire netwerken) of de sterkte/intensiteit ervan (gewogen netwerken). In gerichte netwerken (directed graphs) houdt men bovendien rekening met de richting van de relatie — wat leidt tot onderscheid tussen wederkerige (reciproke) en asymmetrische interacties.

We bespreken hier drie vormen van analyse:

- Dyad census
- Reciprocity
- Assortativity (homophily)

## 4.1. Dyad census

Een dyad census omvat een fundamentele beschrijving van de types relaties die voorkomen tussen alle mogelijke koppels van actoren (dyades) in een gericht netwerk (directed graph). In netwerkanalyse verwijst deze telling naar een classificatie van alle mogelijke dyades binnen het netwerk op basis van hun onderlinge connectiviteit. Concreet worden alle mogelijke paren van actoren gecategoriseerd in drie types:

- Mutual (wederkerige) dyad: Er bestaat een relatie in beide richtingen tussen twee knopen (A $\rightarrow$ B én B $\rightarrow$ A). Dit type wijst op reciprociteit, vaak geassocieerd met sterke of evenwichtige sociale banden.
- Asymmetric dyad: Er is een unilaterale relatie tussen twee knopen (bv. A $\rightarrow$ B, maar niet B $\rightarrow$ A). Dit type kan wijzen op hiërarchie, afhankelijkheid of informatiestromen in één richting.

- Null dyad: Er is geen relatie tussen de twee knopen in eender welke richting (noch A → B, noch B → A). Dit duidt op afwezigheid van interactie of connectie tussen de twee actoren.

Het verkrijgen van een overzicht van de verschillende type dyads kan als volgt (sna package):

```
#detach(package:sna)
# Zet eerst het igraph object om in een matrix die gebruikt kan worden voor het sna packages:
class_matrix <- as.matrix(igraph::as_adjacency_matrix(class_matrix_net, sparse = TRUE))

# Bereken de dyad census
dyad_count <- sna::dyad.census(class_matrix)
dyad_count
```

```
##      Mut Asym Null
## [1,]  19   39  218
```

Dit zegt al iets, maar hoe moeten we deze getallen interpreteren? Zijn ze hoog? Zijn ze laag?

Dat hangt ervan af: Bijvoorbeeld van het aantal actoren – veel actoren betekent veel mogelijke dyades. Of van de dichtheid van het netwerk – in een dun netwerk zijn veel dyads wellicht leeg.

Een natuurlijke manier om de dyad census te interpreteren is door het geobserveerde aantal van de verschillende dyades te vergelijken met het aantal dat we zouden zien in een vergelijkbaar netwerk waarin de verbindingen TOEVALLIG aanwezig of afwezig zijn. Dit kan op de volgende manier in R:

```
#detach(package:sna)
# Bereken enkele centrale kenmerken van het netwerk die we nodig hebben om de random netwerken te simul
net_size <- nrow(class_matrix)
net_dens <- sna::gden(class_matrix)

# Genereer 100 random netwerken op basis van de size en density of het echte netwerk
random_net <- sna::rgraph(net_size, 100, net_dens)

# Klopt de dimensie?
dim(random_net)
```

```
## [1] 100  24  24
```

```
# Plot het random netwerk
#sna::gplot(random_net[1,,],  mode = "kamadakawai")

# Density van het netwerk
random_net_dens <- sna::gden(random_net)

# Dyad census van het netwerk
random_dyad <- sna::dyad.census(random_net)
random_dyad
```

```
##        Mut Asym Null
##   [1,]   7   67  202
##   [2,]  17   62  197
##   [3,]   3   81  192
##   [4,]   8   60  208
```

```
##   [5,]   7   68  201
##   [6,]   7   57  212
##   [7,]   3   64  209
##   [8,]   5   59  212
##   [9,]   8   69  199
##  [10,]   3   63  210
##  [11,]   1   72  203
##  [12,]   5   62  209
##  [13,]   9   70  197
##  [14,]   7   55  214
##  [15,]   8   65  203
##  [16,]   4   76  196
##  [17,]   3   55  218
##  [18,]   2   56  218
##  [19,]   1   69  206
##  [20,]   2   70  204
##  [21,]   3   54  219
##  [22,]  10   74  192
##  [23,]   6   67  203
##  [24,]   8   67  201
##  [25,]   2   65  209
##  [26,]   7   84  185
##  [27,]  10   66  200
##  [28,]   3   73  200
##  [29,]   5   63  208
##  [30,]   5   66  205
##  [31,]   6   67  203
##  [32,]   7   67  202
##  [33,]   5   67  204
##  [34,]   7   65  204
##  [35,]   5   70  201
##  [36,]   4   61  211
##  [37,]   5   65  206
##  [38,]   6   72  198
##  [39,]   7   57  212
##  [40,]   6   63  207
##  [41,]   5   63  208
##  [42,]   6   69  201
##  [43,]   5   63  208
##  [44,]   2   77  197
##  [45,]   3   71  202
##  [46,]   6   60  210
##  [47,]   5   67  204
##  [48,]   4   72  200
##  [49,]   7   70  199
##  [50,]   7   61  208
##  [51,]   4   65  207
##  [52,]   4   59  213
##  [53,]   8   45  223
##  [54,]   6   67  203
##  [55,]   3   72  201
##  [56,]   3   67  206
##  [57,]   6   68  202
##  [58,]   8   55  213
```

```
##  [59,]   3   75  198
##  [60,]   6   57  213
##  [61,]   3   68  205
##  [62,]   5   76  195
##  [63,]   7   62  207
##  [64,]   6   83  187
##  [65,]   2   53  221
##  [66,]   5   71  200
##  [67,]   8   61  207
##  [68,]   0   72  204
##  [69,]   7   74  195
##  [70,]  10   59  207
##  [71,]  11   66  199
##  [72,]   7   61  208
##  [73,]   8   66  202
##  [74,]   7   68  201
##  [75,]   5   63  208
##  [76,]   6   66  204
##  [77,]   3   68  205
##  [78,]   7   69  200
##  [79,]   7   70  199
##  [80,]   5   57  214
##  [81,]   6   73  197
##  [82,]   7   72  197
##  [83,]   8   67  201
##  [84,]   3   75  198
##  [85,]   2   68  206
##  [86,]   4   50  222
##  [87,]   9   61  206
##  [88,]   6   56  214
##  [89,]   6   65  205
##  [90,]   5   76  195
##  [91,]   7   68  201
##  [92,]   7   63  206
##  [93,]   6   67  203
##  [94,]   5   69  202
##  [95,]   6   70  200
##  [96,]   2   70  204
##  [97,]   7   76  193
##  [98,]   3   79  194
##  [99,]   5   57  214
## [100,]  12   61  203
```

Visualiseer de dyad census van het netwerk in vergelijking met de random dyad census:

```
# Install vioplot package
#install.packages("vioplot")
library(vioplot)
```

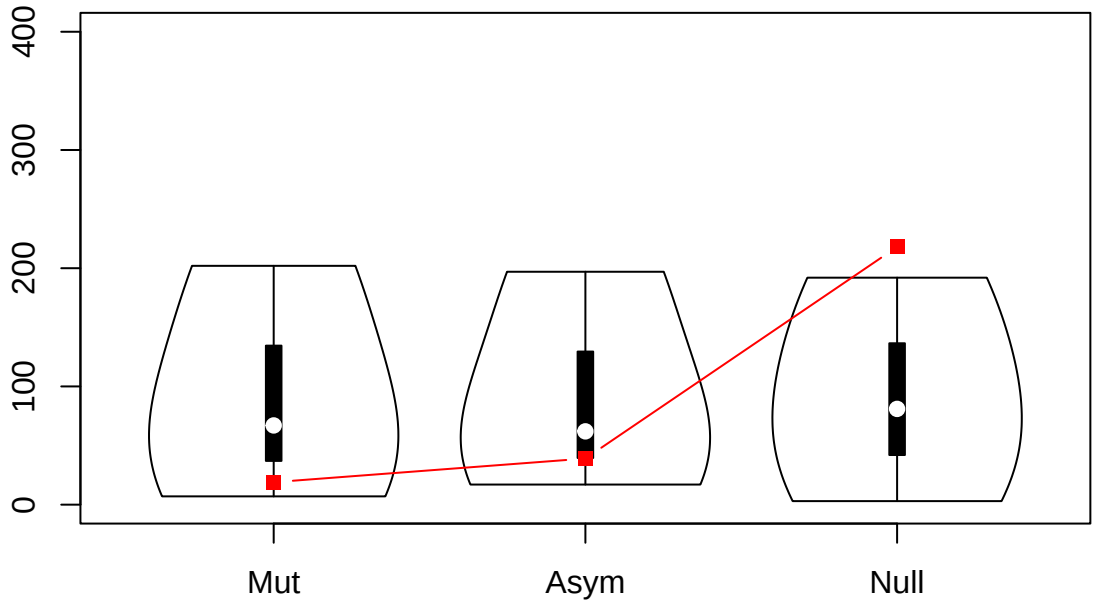
```
## Warning: package 'vioplot' was built under R version 4.3.3
```

```
## Loading required package: sm
```

```
## Warning: package 'sm' was built under R version 4.3.3
```

```
## Package 'sm', version 2.2-6.0: type help(sm) for summary information

## Loading required package: zoo

##
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```r
# Vioplot
vioplot(random_dyad[1,], random_dyad[2,], random_dyad[3,],                # the distribution
        names=colnames(random_dyad)[c(1,2,3)],  # name of the triad type
        col="transparent",                # let the "violin" be transparent
        ylim=c(0, 400))
points(1:3,
       dyad_count[c(1,2,3)],
       col="red",
       type="b",
       pch=15)
```




We zien dat in onze data de wederzijdse dyaden/relaties enigszins ondervertegenwoordigd zijn in vergelijking met de random netwerken. Wat betreft de asymmetrische dyaden/relaties in het vriendschapsnetwerk, zijn deze min of meer gelijk vertegenwoordigd ten opzichte van de random netwerken, maar het gemiddelde aantal

asymmetrische dyaden is kleiner in het vriendschapsnetwerk. Daarentegen zijn de null-dyaden oververtegenwoordigd in onze data in vergelijking met de random netwerken. Dit kan erop wijzen dat er in onze data minder relaties aanwezig zijn dan verwacht zou worden op basis van dezelfde grootte en dichtheid van het netwerk.

## 4.2. Reciprocity

Reciprocity verwijst naar de mate waarin relaties in een gericht netwerk wederzijds zijn, dat wil zeggen, in hoeverre een connectie van actor A naar actor B ook wordt beantwoord door een connectie van B naar A. Het is een fundamenteel concept in sociale netwerkanalyse, vooral bij de bestudering van interacties waarin wederkerigheid van belang is, zoals communicatie, samenwerking of vriendschap.

De mate van reciprocity kan variëren tussen 0 en 1, waarbij 0 betekent dat er geen wederzijdse relaties bestaan en 1 dat alle relaties wederkerig zijn. Een hoge reciprocity duidt op een sterk wederzijds karakter van interacties binnen het netwerk, wat wijst op een intensieve uitwisseling of samenwerking tussen actoren. Een lage reciprocity suggereert daarentegen dat relaties vaak eenzijdig zijn, bijvoorbeeld bij hiërarchische structuren of informatiestromen die niet worden beantwoord.

In het kader van bijvoorbeeld berichtverkeer tussen criminelen, kan een hoge reciprocity wijzen op een stabielere, wederkerige communicatie tussen netwerkleden, terwijl een lage reciprocity eenrichtingsverkeer of asymmetrische relaties kan weerspiegelen:

```
# reciprocity
sna::grecip(class_matrix)
```

```
##       Mut
## 0.8586957
```

```
# deze  lijken niet te kloppen

?grecip
```

```
## starting httpd help server ... done
```

```
sna::grecip(class_matrix, measure = "dyadic.nonnull")
```

```
##       Mut
## 0.3275862
```

```
# = reciprocity die kijkt naar het aantal dyaden waar minstens één relatie aanwezig is (dus dyaden waar
```

`Mut = 0.3275862` met `measure = dyadic.nonnull`: Dit is de reciprociteit gemeten enkel op de dyaden waar minimaal één relatie aanwezig is (de zogenaamde non-null dyads). Dus, van alle dyaden waarin minstens één richting een relatie vertoont, is ongeveer `32,76%` wederkerig (beide kanten hebben een relatie).

`Mut = 0.8586957` zonder extra argument (measure niet ingevuld): Dit is de standaardreciprociteit, meestal berekend over alle dyaden, inclusief dyaden zonder enige relatie (null dyads). Hier is dus de reciprocity ongeveer `85,87%`.

### 4.3. Assortativity mixing (homophily)

Assortativity is een netwerkmaat die de mate van homofilie of heterofilie in een netwerk kwantificeert. Homofilie verwijst naar het fenomeen waarbij nodes met vergelijkbare eigenschappen eerder met elkaar verbonden zijn dan met nodes die verschillen op die eigenschappen. Omgekeerd duidt heterofilie op een voorkeur voor verbindingen tussen nodes met verschillende eigenschappen. In sociale netwerken is assortativity bijvoorbeeld vaak zichtbaar in termen van leeftijd, geslacht of andere demografische kenmerken. Het meten van assortativity geeft inzicht in de structuur en dynamiek van het netwerk, en kan helpen verklaren hoe informatie, gedrag of invloeden zich binnen het netwerk verspreiden.

In het algemeen behelst assortativity of assortativity mixing een maat voor segregatie binnen een (sociaal) netwerk. Het verwijst naar de voorkeur van nodes in een netwerk, in dit geval individuen in een vriendschapsnetwerk, om anderen in het netwerk te bereiken of ermee in contact te komen op basis van verschillende persoonlijke kenmerken/ eigenschappen. Eigenlijk verwijst dit naar de uitdrukking 'beards of feather flock together' " ', wat betekent dat mensen met gelijkaardige eigenschappen of kenmerken geneigd zijn om dezelfde dingen te doen of zich sterker met elkaar lijken te verbinden.

Formeel wordt assortativity berekend als de correlatie tussen eigenschappen van verbonden nodes. Voor numerieke attributen (zoals graad) wordt vaak de Pearson-correlatie gebruikt, terwijl voor categorische attributen (zoals geslacht) de assortativity wordt gemeten als de mate waarin verbindingen vaker voorkomen tussen nodes van dezelfde categorie dan op basis van toeval verwacht zou worden.

Laat ons assortativity berekenen met behulp van het `igraph`-package:

```
# Algemeen:
#igraph::assortativity_degree(class_matrix_net, directed = TRUE)

#Assortativity met gender:
  # Gender attribuut capteren
gender <- igraph::vertex_attr(class_matrix_net, "gender")

  # Voor categorische attributen gebruiken we assortativity_nominal
assortativity_gender <- igraph::assortativity_nominal(class_matrix_net, types = as.factor(gender), dire

  # Print out
print(assortativity_gender)
```

```
## [1] 0.270557
```

# 5. Triad level

Naast dyadische relaties, die de directe connecties tussen twee actoren beschrijven, biedt het triadisch niveau een interessant perspectief op de onderliggende structuur van een netwerk. Een triade bestaat uit een groep van drie nodes (actoren) en de mogelijke relaties tussen hen. De analyse van triads stelt onderzoekers in staat om complexere sociale dynamieken te identificeren, zoals wederkerigheid, geslotenheid van netwerken en structurele spanningen.

Het triadisch niveau vormt daarmee een brug tussen microsociale processen (interacties tussen individuen) en macrosociale netwerkstructuren, en biedt een belangrijk kader om sociale cohesie, hiërarchie, of fragmentatie te analyseren.

Net zoals op het dyadische niveau, kunnen we ook op het triadische niveau een triad census uitvoeren en visualiseren.

Het verkrijgen van een overzicht van de verschillende type triads kan als volgt (sna package):

```
#detach(package:sna)
# Zet eerst het igraph object om in een matrix die gebruikt kan worden voor het sna packages:
#class_matrix <- as.matrix(igraph::as_adjacency_matrix(class_matrix_net, sparse = TRUE))

# Bereken de dyad census
triad_count <- sna::triad.census(class_matrix, mode = 'digraph') #directed; 'graph' voor undirected
triad_count
```

```
##      003 012 102 021D 021U 021C 111D 111U 030T 030C 201 120D 120U 120C 210 300
## [1,] 987 551 277   13   38   31   46   41    8    0  10    4    7    2   6   3
```

Er is ook een functie die toelaat om het type triad te identificeren tussen specifieke nodes:

```
#detach(package:sna)
# Bv tussen actor, 1, 5 en 8
sna::triad.classify(class_matrix, tri=c(1,5,8))
```

```
## [1] "012"
```

```
# of actor 4, 7, and 9?
sna::triad.classify(class_matrix, tri=c(4,7,8))
```

```
## [1] "003"
```

Idem als voor de dyad census kunnen we ook hier de triad census van het netwerk vergelijken met een random triad:

```
#detach(package:sna)
# Bereken enkele centrale kenmerken van het netwerk die we nodig hebben om de random netwerken te simul
net_size <- nrow(class_matrix)
net_dens <- sna::gden(class_matrix)

# Genereer 100 random netwerken op basis van de size en density of het echte netwerk
random_net <- sna::rgraph(net_size, 100, net_dens)

# Klopt de dimensie?
dim(random_net)
```

```
## [1] 100  24  24
```

```
# Plot het random netwerk
#sna::gplot(random_net[1,,],  mode = "kamadakawai")

# Density van het netwerk
random_net_dens <- sna::gden(random_net)

# Dyad census van het netwerk
random_triad <- sna::triad.census(random_net, mode = 'digraph')
random_triad
```

```
##         003 012 102 021D 021U 021C 111D 111U 030T 030C 201 120D 120U 120C 210
##   [1,]  906 784  64   52   51  106   22   18   14    2   1    1    2    1   0
##   [2,]  793 831  62   64   76  128   11   20   16    9   2    4    1    6   1
##   [3,]  701 839  70   75   84  160   26   24   26    8   0    2    2    6   1
##   [4,] 1015 788  13   43   66   76    4    4   13    1   0    1    0    0   0
##   [5,]  775 808  55   75   76  168   13   17   28    7   0    0    0    1   1
##   [6,]  840 776  84   57   58  120   26   29   14    8   2    3    1    5   1
##   [7,]  792 817  78   74   69  116   18   19   23    6   4    1    3    3   1
##   [8,]  801 775  99   70   62  123   34   23   19    3   3    4    3    3   2
##   [9,]  761 810  87   64   78  130   28   21   21    9   2    2    6    4   1
##  [10,]  841 770 105   57   51  101   33   32    8    7   7    2    3    5   2
##  [11,]  876 809  88   40   52   95   12   19   20    3   2    1    1    5   1
##  [12,]  827 813  54   78   75  116   10   15   15   13   1    1    0    6   0
##  [13,]  720 844  53   79   95  139   14   30   36    3   1    2    2    5   1
##  [14,]  793 822  54   77   58  132   21   24   17   17   1    1    2    4   1
##  [15,]  805 747 116   70   60  101   35   32   26    5   5    6    1   10   5
##  [16,]  761 859  44   80   72  140   11    8   29   17   0    1    1    1   0
##  [17,]  850 724 153   42   52   88   41   42    8    4   7    2    4    6   1
##  [18,]  838 796  50   58   75  138   10   23   24    7   0    3    0    2   0
##  [19,]  715 857  66   71   76  142   24   28   27    6   2    1    1    8   0
##  [20,] 1012 740  45   41   45  107    6   13   10    3   0    0    1    1   0
##  [21,]  936 808  32   52   50  107    4    7   19    8   0    0    1    0   0
##  [22,]  871 814  47   65   60  108   12   17   14    5   1    3    3    4   0
##  [23,]  933 731  92   53   41  105   20   19    7    5   3    3    5    5   2
##  [24,]  887 820  27   57   60  133    8    7   20    3   0    1    0    1   0
##  [25,]  818 805  53   81   61  127   24   23   16    8   2    1    3    2   0
##  [26,]  838 825  61   52   65  109   21   20   19    7   1    3    1    2   0
##  [27,]  973 754  34   54   45  108   12   16   20    4   0    0    2    2   0
##  [28,]  730 821  59   70   92  158   18   22   38    7   2    0    4    3   0
##  [29,]  905 818  41   48   55  107   13    8   19    6   0    1    1    2   0
##  [30,]  710 799 102   69   73  157   33   25   31   10   1    3    4    7   0
##  [31,]  840 726 115   52   54  123   38   36   13    5   5    7    1    5   4
##  [32,]  816 772  98   55   59  123   34   27   19    8   2    1    4    4   2
##  [33,] 1077 648  81   48   34   71   15   24   13    6   3    0    0    2   2
##  [34,]  942 761  67   53   56   82   16   19   13    7   0    3    3    2   0
##  [35,]  848 808  57   51   63  120   20   23   17   10   3    3    0    1   0
##  [36,]  727 814  88   73   66  140   40   29   21    9   1    3    5    7   1
##  [37,]  991 797  13   58   46   91    3    5   13    6   0    1    0    0   0
##  [38,]  872 781  97   47   47  112   24   12   14    4   6    1    4    2   1
##  [39,]  681 830  73   75   94  159   34   31   29    6   4    3    2    3   0
##  [40,]  929 708 122   45   40   95   26   24   12    6   9    4    0    4   0
##  [41,]  634 858  75   82   79  167   24   35   40   13   3    3    2    9   0
##  [42,]  875 799  49   53   68  126   15   14   12    4   1    4    2    2   0
##  [43,]  860 848  14   78   54  132    3    5   24    6   0    0    0    0   0
##  [44,]  792 777  87   71   65  127   31   23   26   14   1    2    3    4   1
##  [45,]  778 857  46   72   64  134   18   16   24    8   0    1    1    4   1
##  [46,]  941 803  24   54   56  102    9   10   22    2   0    0    1    0   0
##  [47,]  863 734 120   46   68  101   21   38   16    3   5    5    0    4   0
##  [48,] 1140 688  12   40   46   80    5    3    6    2   0    0    0    2   0
##  [49,]  711 859  64   82   65  150   22   12   36   13   2    2    1    5   0
##  [50,]  749 817 108   60   63  115   29   33   20    7   3    4    5    9   2
##  [51,]  837 834  40   72   70  132   10   11    9    4   0    2    1    2   0
##  [52,]  832 778  76   57   65  119   31   31   13   10   3    0    3    5   1
##  [53,]  915 756  50   61   50  120   17   14   25   10   1    3    1    1   0
```

```
##  [54,]  909 800  71   59   51   77   17   11   17    3   2    4    2    1    0
##  [55,]  968 703 113   35   39   88   18   33   15    3   3    1    1    4    0
##  [56,]  923 745  93   42   57   92   31   17   12    2   2    4    2    1    1
##  [57,]  859 759  77   53   65  117   40   28   18    2   2    0    2    1    1
##  [58,]  778 795  54   77   76  150   17   26   23   16   0    3    1    7    1
##  [59,]  742 815 105   66   58  117   34   36   25    7   1    4    6    5    3
##  [60,]  831 823  40   66   65  138    8   13   26   10   1    0    0    3    0
##  [61,]  703 820  65   81   83  171   28   26   31    5   0    0    1    8    2
##  [62,]  850 776  87   58   52  112   27   27   19    5   2    4    3    2    0
##  [63,]  970 780  32   49   57   83   10   18   16    4   1    1    1    2    0
##  [64,]  869 776  87   61   59   90   28   24   13    5   3    3    2    4    0
##  [65,]  723 816  72   71   83  167   25   19   29    6   2    2    6    2    1
##  [66,]  813 812  65   65   60  137   19   14   19    9   0    1    3    6    1
##  [67,]  894 783  67   69   54  100   25   10   13    3   2    0    2    2    0
##  [68,]  823 840  37   60   56  126   19   25   20   12   1    2    1    2    0
##  [69,]  885 774  81   55   47  110   25   16   17    7   2    2    1    1    1
##  [70,]  851 782  35   84   64  150   13   12   17   10   0    2    2    2    0
##  [71,]  794 809 101   52   61  110   32   27   21    5   4    3    2    3    0
##  [72,]  899 802  27   67   57  126    8    7   17   12   0    1    0    1    0
##  [73,]  758 849  47   71   69  160   17   17   22    7   0    2    2    3    0
##  [74,]  740 795  84   73   70  137   34   41   20   16   1    3    3    5    2
##  [75,]  852 809  37   63   66  135   10   10   25    9   0    1    2    4    1
##  [76,]  846 743  94   68   55  127   23   21   21   14   4    2    0    6    0
##  [77,]  760 754 112   66   58  137   40   37   26   11   6    2    3   10    2
##  [78,]  838 750  98   48   69  121   31   28   23    4   3    1    2    6    2
##  [79,]  869 755  67   57   77  133   16   19   14   10   1    0    4    2    0
##  [80,]  594 834  66   98  110  185   36   35   39   12   2    3    5    5    0
##  [81,]  812 760  82   75   74  120   22   31   23   11   2    1    2    6    3
##  [82,]  810 810  44   76   72  147   16   17   16    6   0    1    2    6    1
##  [83,]  950 710  92   46   49  105   21   31   10    2   2    2    1    3    0
##  [84,]  801 838  34   65   75  151   19    8   24    5   1    1    0    2    0
##  [85,]  848 764  83   56   66  115   24   31   21    3   2    1    0    9    1
##  [86,]  742 787 105   64   67  142   27   41   24    7   5    1    3    7    2
##  [87,]  821 789  67   71   55  137   24   24   15    6   2    2    3    8    0
##  [88,]  787 817  61   78   61  131   28   28   16    4   1    0    3    8    1
##  [89,]  805 817  55   65   81  138   10   13   23    8   0    0    4    4    1
##  [90,]  740 850  43   89   84  158    6   13   26   11   0    1    0    3    0
##  [91,]  893 756  62   69   51  109   20   30   15    5   4    1    3    4    2
##  [92,]  831 816  43   68   73  119   15   23   20   10   1    1    3    1    0
##  [93,]  931 751  81   43   60   88   17   19   15    6   2    2    3    6    0
##  [94,]  748 840  39   83   76  155   16   21   24   13   3    0    4    2    0
##  [95,]  885 851  10   57   53  122    6    5   24   10   0    0    0    1    0
##  [96,] 1033 729  55   46   42   78   16   13    8    0   0    2    2    0    0
##  [97,]  690 853  65   71   71  170   26   21   30   10   3    6    1    7    0
##  [98,]  796 833  46   72   70  142   19   17   16    7   0    2    2    2    0
##  [99,]  769 766 103   66   72  124   50   34    9    9   9    3    2    6    2
## [100,]  886 781  72   54   51  102   25   21   19    3   3    1    4    1    1
##         300
##   [1,]   0
##   [2,]   0
##   [3,]   0
##   [4,]   0
##   [5,]   0
##   [6,]   0
```

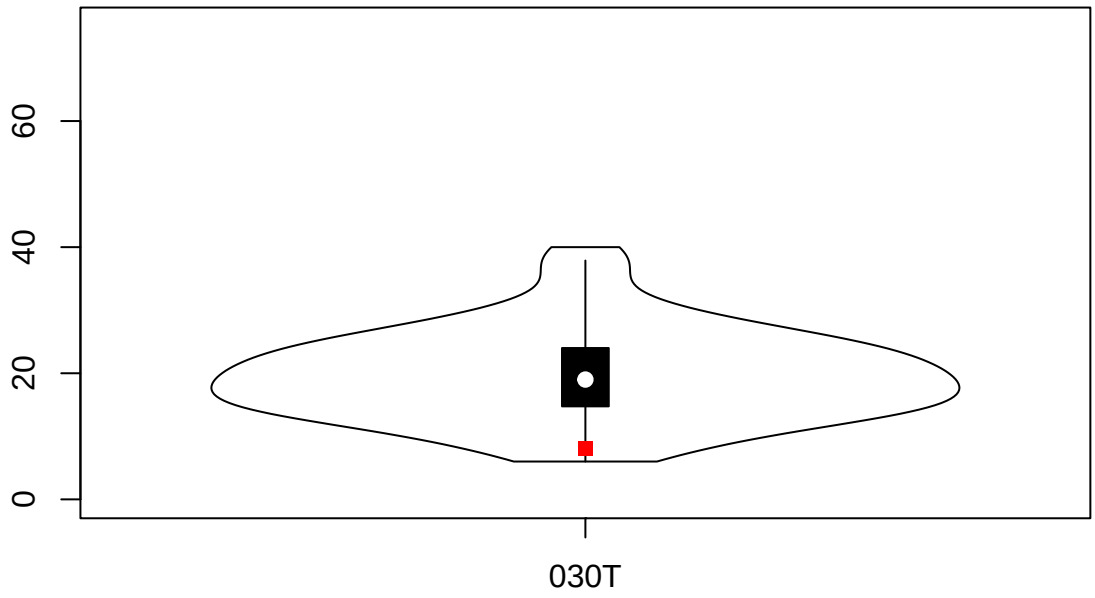
```
##   [7,]   0
##   [8,]   0
##   [9,]   0
##  [10,]   0
##  [11,]   0
##  [12,]   0
##  [13,]   0
##  [14,]   0
##  [15,]   0
##  [16,]   0
##  [17,]   0
##  [18,]   0
##  [19,]   0
##  [20,]   0
##  [21,]   0
##  [22,]   0
##  [23,]   0
##  [24,]   0
##  [25,]   0
##  [26,]   0
##  [27,]   0
##  [28,]   0
##  [29,]   0
##  [30,]   0
##  [31,]   0
##  [32,]   0
##  [33,]   0
##  [34,]   0
##  [35,]   0
##  [36,]   0
##  [37,]   0
##  [38,]   0
##  [39,]   0
##  [40,]   0
##  [41,]   0
##  [42,]   0
##  [43,]   0
##  [44,]   0
##  [45,]   0
##  [46,]   0
##  [47,]   0
##  [48,]   0
##  [49,]   0
##  [50,]   0
##  [51,]   0
##  [52,]   0
##  [53,]   0
##  [54,]   0
##  [55,]   0
##  [56,]   0
##  [57,]   0
##  [58,]   0
##  [59,]   0
##  [60,]   0
```

```
##  [61,]   0
##  [62,]   0
##  [63,]   0
##  [64,]   0
##  [65,]   0
##  [66,]   0
##  [67,]   0
##  [68,]   0
##  [69,]   0
##  [70,]   0
##  [71,]   0
##  [72,]   0
##  [73,]   0
##  [74,]   0
##  [75,]   0
##  [76,]   0
##  [77,]   0
##  [78,]   0
##  [79,]   0
##  [80,]   0
##  [81,]   0
##  [82,]   0
##  [83,]   0
##  [84,]   0
##  [85,]   0
##  [86,]   0
##  [87,]   0
##  [88,]   0
##  [89,]   0
##  [90,]   0
##  [91,]   0
##  [92,]   0
##  [93,]   0
##  [94,]   0
##  [95,]   0
##  [96,]   0
##  [97,]   0
##  [98,]   0
##  [99,]   0
## [100,]   0
```

Alsook op een zelfde manier visualiseren (voorbeeld distributie 030T):

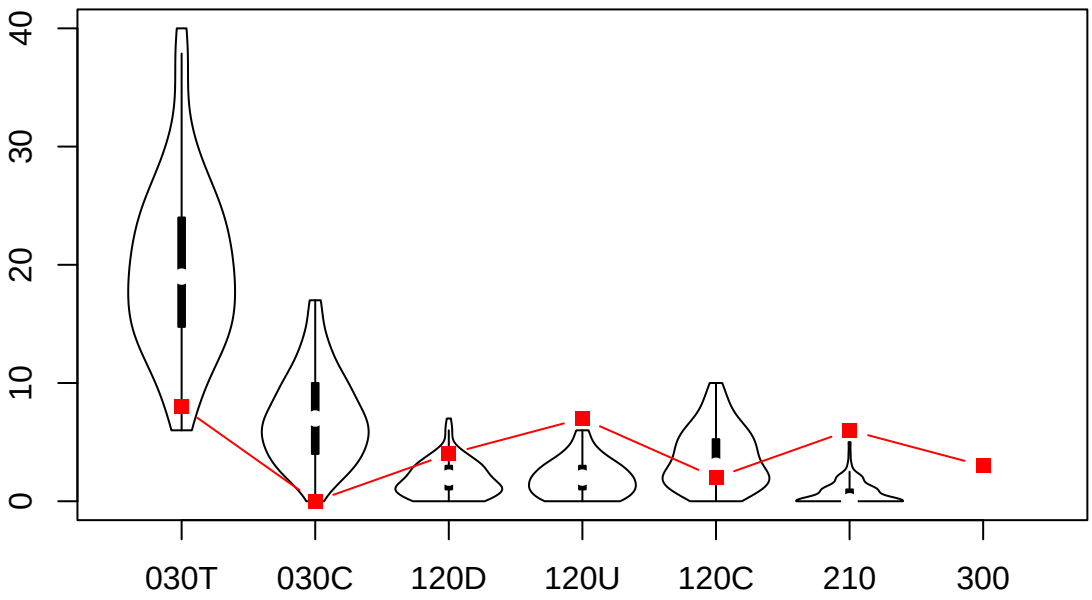
```
# Install vioplot package
#install.packages("vioplot")
#library(vioplot)

# Vioplot: distributie van transitive triplets (030T): gesimuleerd vs echt netwerk
vioplot(random_triad[,9],
        names=colnames(random_triad)[9],
        col="transparent",
        ylim=c(0, 75))

points(1, triad_count[9],
```

```
       col="red",
       pch=15)
```




Of van alle gesloten triads:

```
# Install vioplot package
#install.packages("vioplot")
#library(vioplot)

# closed triangles (in a quite primitive way)
vioplot(random_triad[,9], random_triad[,10], random_triad[,12],
        random_triad[,13], random_triad[,14], random_triad[,15], 0,
        names=colnames(random_triad)[c(9,10,12,13,14,15,16)],
        col="transparent")

points(1:7,
       triad_count[c(9,10,12,13,14,15,16)],
       col="red",
       type="b",
       pch=15)
```




# 6. Tussen-niveau (the in-between): Blockmodelling & community detection

Het tussen-niveau (the in-between) – ook wel het mesoniveau genoemd – biedt een analytisch kader dat zich richt op substructuren die kleiner dan het volledige netwerk. Dit niveau omvat onder meer de studie van triaden, kleine subgroepen, gemeenschappen (clusters of communities) en structurele gaten.

Analyse op dit niveau laat toe om structurele kenmerken van sociale configuraties te detecteren die niet zichtbaar zijn op het micro- of macroniveau, maar die wel cruciaal zijn voor processen zoals informatieverspreiding, sociale controle, en cohesie.

Kenmerkend voor het tussen-niveau is de focus op lokale patronen van interactie: wie vormt bruggen tussen clusters? Wat zegt dit over wederzijds vertrouwen of groepsvorming? Welke actoren fungeren als brokers of gatekeepers tussen anders gescheiden delen van het netwerk?

We zullen hier ingaan op een aantal belangrijke methoden en analyses op dit niveau, met name: - Blockmodelling en structurele equivalentie - Cliques en community detectie

## 6.1. Blockmodelling

Blockmodelling is een analysetechniek binnen de sociale netwerkanalyse die tot doel heeft structuurpatronen te identificeren tussen groepen van actoren. In plaats van het netwerk op het niveau van individuele nodes te bestuderen, groepeert blockmodelling actoren op basis van gelijkaardigheid in hun relaties met anderen. Deze gegroepeerde actoren worden geordend in zogenaamde blokken, waarbij men kijkt naar hoe deze blokken

onderling verbonden zijn.Het concept 'equivalence' staat hierbij centraal: equivalence drukt uit in welke mate actoren gelijken op elkaar in termen van met wie ze verbonden zijn in een netwerk.

Het fundamentele uitgangspunt is dat actoren die op gelijkaardige wijze met andere actoren verbonden zijn, mogelijks ook een gelijkaardige sociale rol vervullen. Door deze structurele equivalentie of bredere rol-equivalentie bloot te leggen, kunnen meer complexe netwerken worden geherstructureerd tot een eenvoudiger blokmatrix die de globale architectuur van het netwerk weerspiegelt.

Hoe structurele gelijkheid meten? Om actoren te groeperen op basis van hun structurele gelijkenissen, moeten deze eerst worden gekwantificeerd. Hiervoor bestaan verschillende maatstaven, waaronder:

- Jaccard-index: vergelijkt de verhouding van gedeelde verbindingen t.o.v. het totaal aantal unieke verbindingen;
- Hamming distance: telt het aantal verschillen tussen de connectiepatronen van twee actoren;
- Simple matching coefficient: vergelijkt zowel overeenkomende connecties als overeenkomende niet-connecties;
- Correlatiecoëfficiënten (zoals Pearson of Spearman): meten de correlatie tussen de verbindingsvectoren van twee actoren.

Deze maten bieden verschillende invalshoeken om equivalentie of rolgelijkenis te definiëren, afhankelijk van het theoretisch kader of het type netwerk (bijv. binair vs. gewogen, gericht vs. ongericht).

- Laat ons om te beginnen nog even ons netwerk snel plotten:

```
# Plot the matrix:
#plot <- sna::gplot(class_matrix)
#sna::gplot(class_matrix, coord = plot)
```

- We clusteren vervolgens de verschillende actoren op basis van een specifieke distance maat, we kiezen hier voor de hamming distance

```
# We berekenen de equivalentie (let op: er bestaan verschillende cluster methoden en distance maten)
library(sna)
```

```
## Warning: package 'sna' was built under R version 4.3.3
```

```
## Loading required package: statnet.common
```

```
## Warning: package 'statnet.common' was built under R version 4.3.3
```

```
##
## Attaching package: 'statnet.common'
```

```
## The following objects are masked from 'package:base':
##
##     attr, order
```

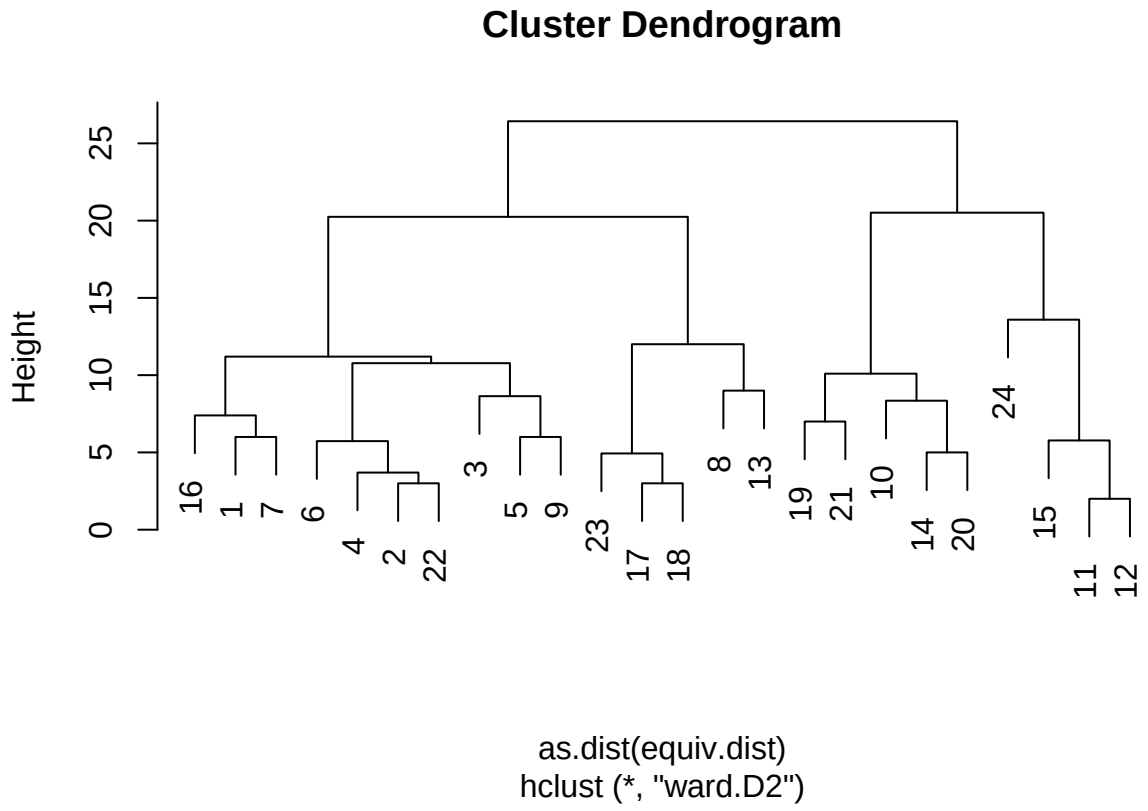
```
## Loading required package: network
```

```
## Warning: package 'network' was built under R version 4.3.3
```

```
##
## 'network' 1.19.0 (2024-12-08), part of the Statnet Project
## * 'news(package="network")' for changes since last version
## * 'citation("network")' for citation information
## * 'https://statnet.org' for help, support, and other information
```

```
## sna: Tools for Social Network Analysis
## Version 2.8 created on 2024-09-07.
## copyright (c) 2005, Carter T. Butts, University of California-Irvine
##  For citation information, type citation("sna").
##  Type help(package="sna") to get started.
```

```
equiv <- sna::equiv.clust(class_matrix, cluster.method="ward.D2", method="hamming")
plot(equiv)
```



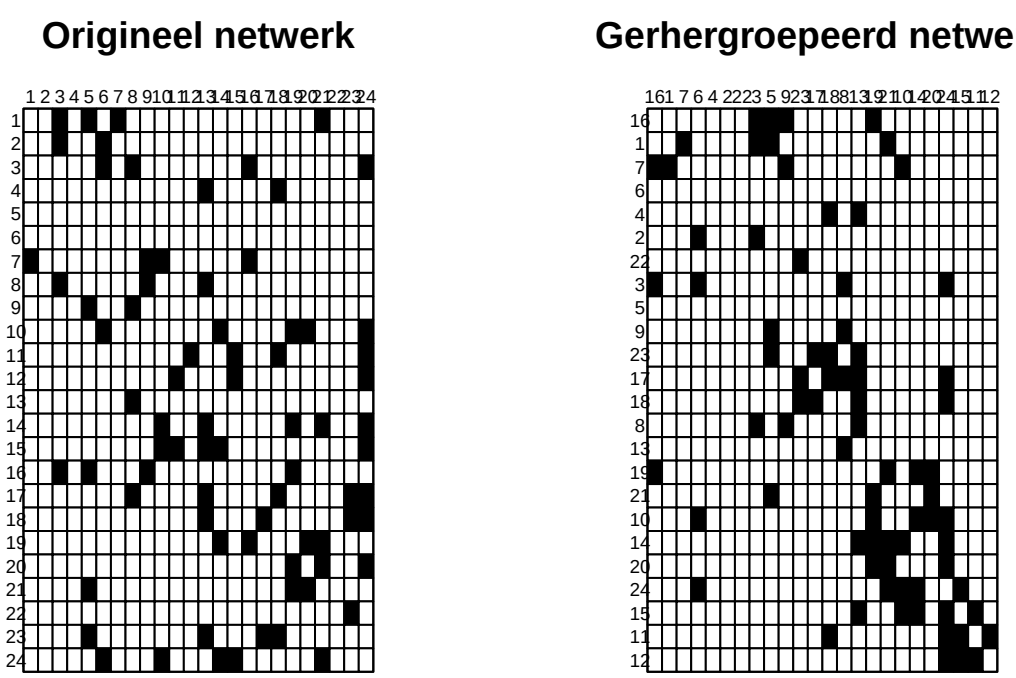

De bovenstaande figuur toont een hiërarchisch cluster-dendrogram, gebaseerd op structurele equivalentie tussen actoren in het netwerk. Door gebruik te maken van de `Ward.D2` clustermethode werden de afstanden tussen actoren geaggregeerd op een wijze die interne clusterhomogeniteit maximaliseert en variantie tussen clusters minimaliseert.

Bij visuele inspectie van het dendrogram is er een duidelijke sprong waarneembaar in de verticale as (hoogte) tussen ~10 en ~20. Deze sprong duidt op een natuurlijke breuk in de datahiërarchie en suggereert een geschikt niveau om het dendrogram af te snijden. Wanneer een horizontale lijn wordt getrokken op ongeveer hoogte

12 à 13, ontstaan er vier duidelijke clusters. Dit betekent dat de 24 nodes in het netwerk het best gegroepeerd kunnen worden in vier blokken.

In de context van blockmodelling impliceert dit dat actoren binnen elk van deze vier clusters equivalent zijn. Met andere woorden, zij hebben gelijkaardige patronen van relaties met andere nodes binnen het netwerk. Dit aantal, met name 4 clusters, dienen we nu als input te gebruiken in R om een simpel blockmodel te maken:

```
# De blockmodel() functie 'cuts the tree', maar hergroepeert ook de actoren in de nieuwe blokken
bm <- sna::blockmodel(class_matrix, equiv, k=4)

# Plot
par(mfrow = c(1, 2), mar = c(4, 4, 2, 2))  # pas marges aan indien nodig
plot.sociomatrix(class_matrix, diaglab = FALSE, cex = 0.6, main = "Origineel netwerk")
plot.sociomatrix(bm$blocked.data, diaglab = FALSE, cex = 0.6, main = "Gerhergroepeerd netwerk")
```




Deze sociomatrices tonen de structuur van relaties in het netwerk, met links het originele netwerk en rechts het gerhergroepeerde netwerk op basis van blockmodelling. De zwarte vlakken geven de aanwezigheid van een relatie tussen twee actoren weer (bijvoorbeeld: persoon i heeft een relatie met persoon j), terwijl witte vlakken afwezigheid aanduiden.

- Origineel netwerk (links): De zwarte vlakken zijn verspreid zonder duidelijke structuur. De matrix toont weinig clustering of patroonvorming: het netwerk lijkt op het eerste zicht ongeorganiseerd. Actoren staan op willekeurige volgorde, waardoor onderliggende structuur moeilijk te herkennen is.
- Gerhergroepeerd netwerk (rechts): Hier zijn de rijen en kolommen geherordend op basis van structurele of regelmatige equivalentie via blockmodelling. De zwarte blokken clusteren nu zichtbaar samen, wat

wijst op onderliggende groepen of posities van actoren met gelijkaardig verbindingsgedrag. Zwarte blokken langs de diagonaal geven aan dat bepaalde clusters veel intra-clusterrelaties vertonen. Witte zones tussen de zwarte blokken suggereren een gebrek aan verbinding tussen bepaalde clusters, wat kan duiden op structuren zoals hiërarchie, segmentatie of segregatie in het netwerk.

We kunnen vervolgens ook het netwerk opnieuw plotten met de nodes/actoren ingekleurd volgens 'blok lidmaaschap' (basic plot):

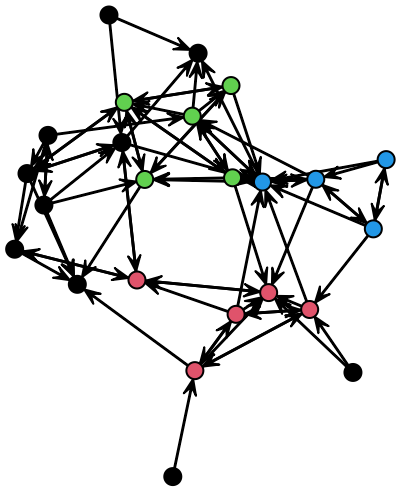
```
# hH haal je de bloklidmaatschappen eruit?
str(bm, 1)
```

```
## List of 11
##  $ block.membership: int [1:24] 1 1 1 1 1 1 1 1 1 1 ...
##  $ order.vector    : int [1:24] 16 1 7 6 4 2 22 3 5 9 ...
##  $ block.content   : chr "density"
##  $ blocked.data    : num [1:24, 1:24] 0 0 1 0 0 0 0 1 0 0 ...
##   ..- attr(*, "dimnames")=List of 2
##  $ block.model     : num [1:4, 1:4] 0.156 0.06 0.06 0.025 0.1 ...
##   ..- attr(*, "dimnames")=List of 2
##  $ plabels         : chr [1:24] "16" "1" "7" "6" ...
##  $ glabels         : chr [1:24] "1" "2" "3" "4" ...
##  $ rlabels         : chr [1:4] "Block 1" "Block 2" "Block 3" "Block 4"
##  $ cluster.method  : chr "ward.D2"
##  $ equiv.fun       : chr "sedist"
##  $ equiv.metric    : chr "hamming"
##  - attr(*, "class")= chr "blockmodel"
```

```
# waarschijnlijk via bm.w3$block.membership
# maar opgelet - ze staan in de verkeerde volgorde!
bm$order.vector
```

```
##  [1] 16  1  7  6  4  2 22  3  5  9 23 17 18  8 13 19 21 10 14 20 24 15 11 12
```

```
# je kunt de bloklidmaatschappen in de juiste volgorde krijgen met deze regel
block_members <- bm$block.membership[order(bm$order.vector)]

# we gebruiken nu gplot met de correcte kleur per knoop op basis van bloklidmaatschap
block_plot <- gplot(class_matrix, vertex.col=block_members)
```



```
block_plot
```

```
##                   x           y
##  [1,] -12.17553268  -4.16186812
##  [2,]  -9.87325335   2.56288427
##  [3,]  -9.41745959  -1.94894774
##  [4,]  -1.24092766 -10.08500424
##  [5,] -10.99331967  -6.96647954
##  [6,]  -6.72014425   1.20990782
##  [7,] -12.03076610  -1.69659544
##  [8,]  -8.88300998  -6.81305614
##  [9,] -13.21015384  -5.72353453
## [10,]  -6.92574943  -1.01544707
## [11,]  -0.51952779  -5.00389617
## [12,]  -0.06674012  -2.55417229
## [13,]  -4.22364029  -7.25907922
## [14,]  -5.50099539  -3.19703143
## [15,]  -2.55600183  -3.24897417
## [16,] -12.77342217  -3.04761542
## [17,]  -5.37564837  -8.02779998
## [18,]  -2.77263598  -7.85665681
## [19,]  -9.33027098  -0.52675719
## [20,]  -5.55518712   0.06680496
## [21,]  -8.60868864  -3.25564111
## [22,]  -7.61525760 -13.76895572
## [23,]  -6.83406435 -10.02016278
```

```
## [24,]  -4.44232939  -3.34683394
```

```
# tadaa! en dit is waarvoor blockmodelling nuttig is...
```

Visueel valt op dat:

- Binnen-blokconnectiviteit: Sommige clusters van nodes (met dezelfde kleur) vertonen een relatief hoge interne densiteit. Dit wijst op blokken waarin actoren voornamelijk onderling relaties aangaan, wat kan duiden op hechte subgroepen of vriendengroepen binnen het netwerk.
- Tussen-blokrelaties: In andere gevallen lijken de pijlen voornamelijk gericht naar actoren buiten het eigen blok. Dit suggereert een rolverdeling waarbij bepaalde groepen vooral relaties aangaan met externe actoren. Dergelijke patronen kunnen wijzen op asymmetrische rolstructuren, zoals bemiddelaars of brugfiguren tussen subgroepen.
- Structuur en hiërarchie: Het netwerk toont geen volledig willekeurige verspreiding van relaties, maar eerder een georganiseerde structuur waarbij enkele blokken centrale posities innemen met veel inkomende of uitgaande verbindingen. Dit zou kunnen wijzen op hiërarchische dynamieken of de aanwezigheid van centrale actoren in termen van informatie- of invloedstromen.
- Functionele differentiatie: Door actoren te groeperen op basis van hun netwerkpositie, eerder dan op inhoudelijke kenmerken (zoals geslacht of leeftijd), wordt zichtbaar hoe bepaalde rollen binnen het netwerk verdeeld zijn. Actoren in hetzelfde blok vervullen vermoedelijk een vergelijkbare sociale functie binnen het geheel.

Maar dit is eigenlijk een klein beetje unfair... vermits we eigenlijk zouden moeten focussen op de centrale componenten van het netwerk. Wanneer we blockmodelling uitvoeren op het volledige netwerk, inclusief geïsoleerde knopen of kleine perifere clusters die nauwelijks verbonden zijn met de rest van het netwerk, dan introduceren we ruis en mogelijk vertekende structuurpatronen. Dit is om verschillende redenen problematisch:

- Focussen op structurele patronen vereist connectiviteit: Blockmodelling is ontworpen om structurele of stochastische equivalentie te detecteren. Deze concepten veronderstellen dat actoren zich in een gemeenschappelijk, verbonden systeem bevinden waarin ze gelijkaardige posities innemen ten opzichte van anderen. Actoren die slechts één connectie hebben of volledig geïsoleerd zijn, dragen nauwelijks bij aan het blootleggen van dergelijke patronen en kunnen de modelresultaten vervormen.
- Perifere knopen verstoren clustering: In hiërarchische clustering (zoals zichtbaar in de dendrogrammen) worden perifere of zwak verbonden knopen vaak foutief gegroepeerd op basis van minimale afstandscriteria. Hierdoor kunnen blokken ontstaan die methodologisch correct zijn, maar inhoudelijk weinig betekenis hebben.
- Centrale componenten zijn representatiever voor groepsstructuur: In sociale netwerken bevindt de belangrijkste structurele informatie zich meestal in de grootste verbonden component. Deze component bevat de meeste interactie, uitwisseling en potentieel voor invloed. Analyse van deze centrale component laat toe om relevantere structurele inzichten te bekomen, bijvoorbeeld over subgroepen, rolverdeling of diffusiekanalen.
- Vergelijking met random netwerken vereist gelijkwaardige structuur: Bij simulaties of vergelijkingen met random netwerken (bv. bij triad census of density-analyses), moet je ervoor zorgen dat je vergelijkt met een netwerk met een vergelijkbare structuur. Het opnemen van losstaande actoren verlaagt artificieel de densiteit, mutualiteit of clustering in het netwerk — wat de vergelijkingsbasis oneerlijk maakt.

Laat ons daarom even deze stap herhalen op de belangrijkste component (main component) van het netwerk:

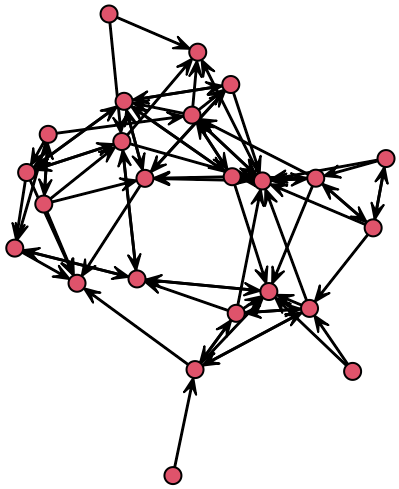
```
# Laten we ons nu richten op de grootste component van het netwerk
# Indien je wil nagaan of het netwerk sterk of zwak geconnecteerd is:
is.connected(class_matrix, connected = "strong")  # voor sterke componenten
```

```
## [1] FALSE
```

```
# Wat betekent "zwakke" verbondenheid?
# Bij zwakke verbondenheid worden richtingen van relaties genegeerd: twee knopen zijn zwak verbonden al
# een pad (in eender welke richting) bestaat tussen hen. Bij sterke verbondenheid moet er een gericht p
# bestaan van knoop A naar B én van B naar A.

# Laat ons nu de grootste component capteren
cfriend <- sna::component.largest(class_matrix, connected = "weak", result = "graph")
cfriend
```

```
##    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
## 1  0 0 1 0 1 0 1 0 0  0  0  0  0  0  0  0  0  0  0  1  0  0  0
## 2  0 0 1 0 0 1 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
## 3  0 0 0 0 0 1 0 1 0  0  0  0  0  0  0  1  0  0  0  0  0  0  1
## 4  0 0 0 0 0 0 0 0 0  0  0  0  1  0  0  0  0  1  0  0  0  0  0
## 5  0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
## 6  0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
## 7  1 0 0 0 0 0 0 0 1  1  0  0  0  0  0  1  0  0  0  0  0  0  0
## 8  0 0 1 0 0 0 0 0 1  0  0  0  1  0  0  0  0  0  0  0  0  0  0
## 9  0 0 0 0 1 0 0 1 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
## 10 0 0 0 0 0 1 0 0 0  0  0  0  0  1  0  0  0  0  1  1  0  0  0  1
## 11 0 0 0 0 0 0 0 0 0  0  0  1  0  0  1  0  0  1  0  0  0  0  0  1
## 12 0 0 0 0 0 0 0 0 0  0  1  0  0  0  1  0  0  0  0  0  0  0  0  1
## 13 0 0 0 0 0 0 0 1 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
## 14 0 0 0 0 0 0 0 0 0  1  0  0  1  0  0  0  0  0  1  0  1  0  0  1
## 15 0 0 0 0 0 0 0 0 0  1  1  0  1  1  0  0  0  0  0  0  0  0  0  1
## 16 0 0 1 0 1 0 0 0 1  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0
## 17 0 0 0 0 0 0 0 1 0  0  0  0  1  0  0  0  0  1  0  0  0  0  1  1
## 18 0 0 0 0 0 0 0 0 0  0  0  0  1  0  0  0  1  0  0  0  0  0  1  1
## 19 0 0 0 0 0 0 0 0 0  0  0  0  0  1  0  1  0  0  0  1  1  0  0  0
## 20 0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  1  0  1  0  0  1
## 21 0 0 0 0 1 0 0 0 0  0  0  0  0  0  0  0  0  0  1  1  0  0  0  0
## 22 0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0
## 23 0 0 0 0 1 0 0 0 0  0  0  0  1  0  0  0  1  1  0  0  0  0  0  0
## 24 0 0 0 0 0 1 0 0 0  1  0  0  0  1  1  0  0  0  0  0  1  0  0  0
```

```
# De grootste component ziet er als volgt uit
gplot(cfriend, coord = block_plot)
```



Blijkbaar is er dus maar 1 grote component, met name het volledige netwerk. . . , geen nood dus hier om de blockmodelling te herhalen, maar goed dat we dit even nagaan.

Wat betekent dit concreet?

- Zwakke connectiviteit beschouwt alleen het bestaan van een pad tussen twee nodes, ongeacht de richting van de relaties. Dit is vaak relevant in sociale netwerken waarbij het belangrijker is dát er een verbinding is, dan wie de relatie 'begon'.
- Dat het gehele netwerk één component vormt, suggereert dat er geen geïsoleerde subgroepen zijn – iedereen is op de een of andere manier met iedereen verbonden.
- Voor sterke connectiviteit (waar de richting van relaties wel telt) kan dit anders liggen. Als het netwerk niet sterk geconnecteerd is, betekent dit dat je niet van elke actor naar elke andere actor kan navigeren via gerichte paden.

## 6.2. Cliques en communities

In dit deel analyseren we de onderliggende structuur van sociale netwerken via cliques, communities en k-cores.

Een clique is een volledig verbonden subgroep waarin elke actor een directe relatie heeft met elke andere actor – dit wijst op maximale cohesie.

Communities zijn minder strikt: ze groeperen actoren die sterker met elkaar verbonden zijn dan met de rest van het netwerk. Ze geven inzicht in sociale clustering en groepsvorming.

K-cores ten slotte identificeren delen van het netwerk waarin elke actor minstens k verbindingen heeft met andere leden van diezelfde groep. Deze methode onthult de kernstructuur van een netwerk en laat zien wie het best ingebed is.

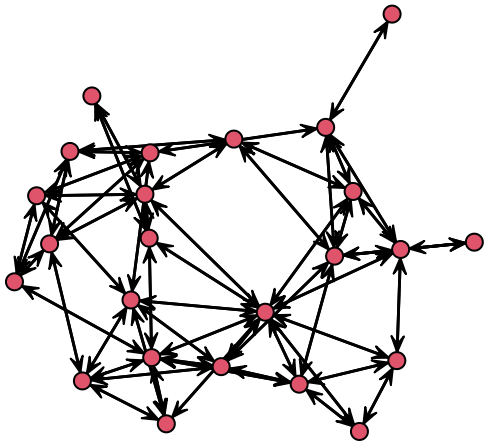
We doen hiervoor beroep op het `igraph`-package. Het 'nadeel' hier is dat deze methoden/functies enkel toepasbaar zijn op undirected netwerken, waardoor we voor deze analyses het netwerk zullen moeten omzetten naar een undirected netwerk alsook naar een `igraph`-object.

- We maken het netwerk eerst undirected, wat in twee stappen kan gedaan worden:

```
# We transposen het netwerk om het undirected te maken:
class_und <- class_matrix + t(class_matrix ) # check table(friend.w1) to see what happened
class_und[class_und==2] <- 1
class_und # Print
```

```
##    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
## 1  0 0 1 0 1 0 1 0 0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0
## 2  0 0 1 0 0 1 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
## 3  1 1 0 0 0 1 0 1 0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  1
## 4  0 0 0 0 0 0 0 0 0  0  0  0  1  0  0  0  0  1  0  0  0  0  0  0
## 5  1 0 0 0 0 0 0 0 1  0  0  0  0  0  0  1  0  0  0  0  1  0  1  0
## 6  0 1 1 0 0 0 0 0 0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  1
## 7  1 0 0 0 0 0 0 0 1  1  0  0  0  0  0  1  0  0  0  0  0  0  0  0
## 8  0 0 1 0 0 0 0 0 1  0  0  0  1  0  0  0  1  0  0  0  0  0  0  0
## 9  0 0 0 0 1 0 1 1 0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0
## 10 0 0 0 0 0 0 1 1 0 0  0  0  0  1  1  0  0  0  1  1  0  0  0  1
## 11 0 0 0 0 0 0 0 0 0 0  0  1  0  0  1  0  0  1  0  0  0  0  0  1
## 12 0 0 0 0 0 0 0 0 0 0  0  1  0  0  0  1  0  0  0  0  0  0  0  1
## 13 0 0 0 1 0 0 0 1 0  0  0  0  0  1  1  0  1  1  0  0  0  0  1  0
## 14 0 0 0 0 0 0 0 0 0 0  1  0  0  1  0  1  0  0  0  1  0  1  0  0  1
## 15 0 0 0 0 0 0 0 0 0 0  1  1  1  1  1  0  0  0  0  0  0  0  0  0  1
## 16 0 0 1 0 1 0 1 0 1  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0
## 17 0 0 0 0 0 0 0 0 1 0  0  0  0  1  0  0  0  0  1  0  0  0  0  1  1
## 18 0 0 0 1 0 0 0 0 0  0  1  0  1  0  0  0  1  0  0  0  0  0  1  1
## 19 0 0 0 0 0 0 0 0 0 0  1  0  0  0  1  0  1  0  0  0  1  1  0  0  0
## 20 0 0 0 0 0 0 0 0 0 0  1  0  0  0  0  0  0  0  0  1  0  1  0  0  1
## 21 1 0 0 0 1 0 0 0 0  0  0  0  0  1  0  0  0  0  1  1  0  0  0  1
## 22 0 0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  1  0
## 23 0 0 0 0 1 0 0 0 0  0  0  0  1  0  0  0  1  1  0  0  0  1  0  0
## 24 0 0 1 0 0 1 0 0 0  1  1  1  0  1  1  0  1  1  0  1  1  0  0  0
```

```
gplot(class_und) # Plot
```



Het 'small-world' fenomeen: het wordt vaak vastgesteld dat netwerken in de echte wereld – vooral grote netwerken – relatief korte gemiddelde padlengtes hebben en een hoge clustering vertonen (veel gesloten triaden). Als gevolg daarvan kunnen de meeste knopen in het netwerk met slechts enkele stappen worden bereikt – het is dus een kleine wereld.

Hoe klein is onze geanalyseerde gemeenschap in dit opzicht?

```
# We transformeren het netwerk eerst naar een igraph object
class_graph <- igraph::graph_from_adjacency_matrix(class_matrix)
class_graph <- igraph::as.undirected(class_graph)
```

```
## Warning: 'as.undirected()' was deprecated in igraph 2.1.0.
## i Please use 'as_undirected()' instead.
## This warning is displayed once every 8 hours.
## Call 'lifecycle::last_lifecycle_warnings()' to see where this warning was
## generated.
```

```
# is het netwerk geconnecteerd?
igraph::is_connected(class_graph)
```

```
## [1] TRUE
```

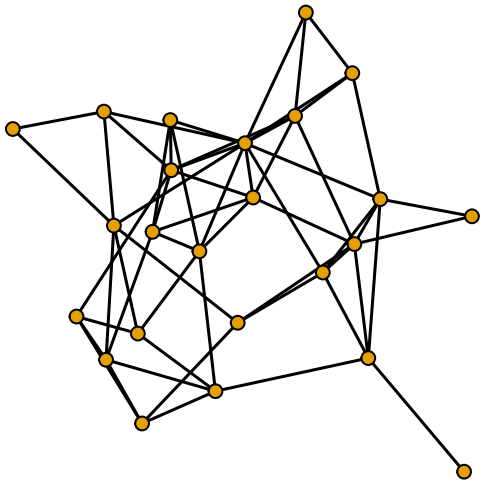
```
# Hoeveel componenten zijn er van verschillende grootte?
components <- igraph::decompose(class_graph)
igraph::vcount(components[[1]])
```

```
## [1] 24
```

```
# Tel het aantal nodes in een component en plot een frequentie tabel
table(sapply(components, igraph::vcount))
```

```
##
## 24
##  1
```

```
# the main component contains two thirds of the vertices (33 of them)

# Bekijk het netwerk opnieuw even om de componenten na te gaan
myLayout <- igraph::layout_with_fr(class_graph)
plot(class_graph,
     edge.color = "black",
     edge.width = 1.5,
     edge.arrow.size = 0.25,
     vertex.size = 6,
     vertex.label = "",
     layout=myLayout)
```



```
# laat ons de centrale component opslaan in één object
main_component <- components[[1]]
```

```r
# sla ook de 'members op' van het component
main_component_members <- as.numeric(substr(names(igraph::V(main_component)),2,3))

# is het een small world?
igraph::mean_distance(main_component) # average path length
```

```
## [1] 2.188406
```

```r
igraph::diameter(main_component) # length of longest path
```

```
## [1] 5
```

```r
igraph::transitivity(main_component) # ratio of closed to open triads => relatively high clustering in
```

```
## [1] 0.3345725
```

**Wat denken jullie? Is het een 'small world'?**

Het geanalyseerde netwerk vertoont duidelijke enkele kenmerken van een small-world structuur. De gemiddelde padlengte bedraagt 2.19 stappen, wat wijst op een hoge mate van bereikbaarheid tussen de actoren. Daarnaast is de clusteringcoëfficiënt 0.33, wat betekent dat ongeveer een derde van de potentiële triaden gesloten zijn. Dit duidt op een aanzienlijke mate van lokale cohesie. De diameter van het netwerk bedraagt slechts 5, wat bevestigt dat de verste actoren binnen een relatief kort pad met elkaar verbonden zijn. Samen ondersteunen deze bevindingen de hypothese dat het netwerk een small-world topologie vertoont, al zouden we dit moeten vergelijken met een random netwerk op basis van de structuur van het echte netwerk...

Hoe dan ook: Onafhankelijk van het feit dat het netwerk een 'small world' is of niet, kunnen we gaan kijken naar de interne structuren van het netwerk...

- Hoeveel cliques zijn er in het netwerk?

```r
# Cliques
cliques <- igraph::cliques(class_graph)
length(cliques) # dit lijkt een lang object - hoe komt dat?
```

```
## [1] 115
```

- Elke clique kan namelijk ook een deelverzameling zijn van een grotere clique. Daarom kan het interessanter zijn om het aantal cliques te tellen:

```r
#table(sapply(igraph::max_cliques(class_graph), length))
all_cliques <- igraph::cliques(class_graph)
clique_sizes <- sapply(all_cliques, length)
table(clique_sizes)
```

```
## clique_sizes
##  1  2  3  4
## 24 58 30  3
```

- Of om het aantal cliques te tellen: Een maximale clique is een volledig verbonden subgroep van knopen die niet meer vergroot kan worden zonder de volledige verbondenheid te verliezen.

```r
table(sapply(igraph::max_cliques(class_graph), length))
```

```
##
##  2  3  4
##  9 18  3
```

- Hoeveel maximale cliques zijn er? En hoe groot is elke clique (bonus):

```r
max_cliques <- igraph::max_cliques(class_graph)
length(max_cliques)
```

```
## [1] 30
```

```r
clique_sizes <- sapply(max_cliques, length)
min(clique_sizes)  # kleinste clique
```

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

```r
max(clique_sizes)  # grootste clique
```

```
## [1] 4
```

Hoewel cliques een intuïtieve manier bieden om sterk verbonden subgroepen in netwerken te identificeren, is de definitie ervan vaak te strikt. Om ook minder perfecte maar nog steeds sterke subgroepen te detecteren, kunnen we gebruik maken van het concept van **k-cores**.

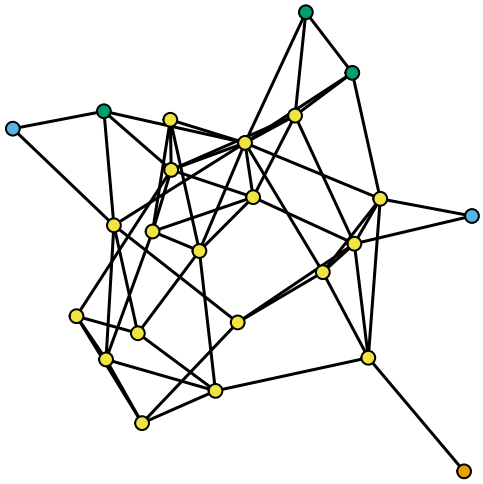
Een *k-core* is een subgroep in een netwerk waarin elke actor minstens $k$ connecties heeft **binnen diezelfde subgroep**. In andere woorden: als we het netwerk beperken tot de nodes in een k-core, dan heeft elke actor daarin minstens k buren. Deze methode laat toe om **cohesieve clusters** te vinden, ook wanneer er geen volledige verbondenheid is zoals in cliques.

We kunnen voor elke node in het netwerk berekenen tot welk k-core niveau hij behoort, en deze vervolgens visualiseren via een kleurenplot.

```r
# k-core scores berekenen voor elke knoop
cores <- igraph::coreness(class_graph)
cores
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
##  4  2  4  2  4  3  4  4  4  4  3  3  4  4  4  4  4  4  4  4  4  1  4  4
```

```r
# Visualisatie van het netwerk met kleuren op basis van k-core lidmaatschap
plot(class_graph,
     edge.color = "black",
     edge.width = 1.5,
     edge.arrow.size = 0.25,
     vertex.size = 6,
     vertex.label = "",
     vertex.color = cores,
     layout = myLayout,
     main = 'k-cores')
```




De meeste nodes zitten in de 4-core, wat betekent dat ze behoren tot een subgroep waarin iedereen minstens vier connecties heeft binnen die groep.

Slechts één actor (actor 22) zit in de 1-core: deze actor is het minst ingebed en behoort tot geen enkele sterk verbonden substructuur hoger dan k=1.

Enkele actoren zitten in lagere cores zoals de 2-core (actor 2, 4) of 3-core (actor 6, 11, 12).

Een andere populaire manier om de structuur van netwerken te verkennen, is via het detecteren van **communities**.

Communitydetectiealgoritmes proberen regio's binnen het netwerk te identificeren die **sterk onderling verbonden zijn**, maar **zwak verbonden zijn met andere delen** van het netwerk. Dit is vaak een niet-triviale optimalisatie-oefening, zeker in grotere of complexere netwerken.

We kunnen bijvoorbeeld het **fast-greedy communitydetectiealgoritme** toepassen op ons netwerk:

```
# Pas communitydetectie toe
communities <- igraph::cluster_fast_greedy(class_graph)

# Aantal gedetecteerde communities
length(communities)
```

```
## [1] 4
```

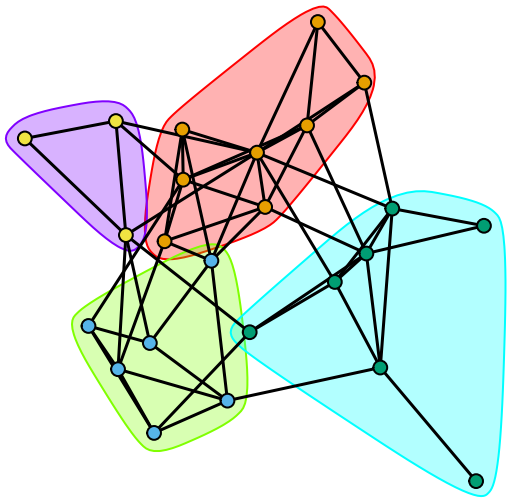
```
# Grootte van elke community
igraph::sizes(communities)
```

```
## Community sizes
## 1 2 3 4
## 8 6 7 3
```

```
# Lidmaatschap van elke node
igraph::membership(communities)
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
##  2  4  4  3  2  4  2  3  2  1  1  1  3  1  1  2  3  3  1  1  2  3  3  1
```

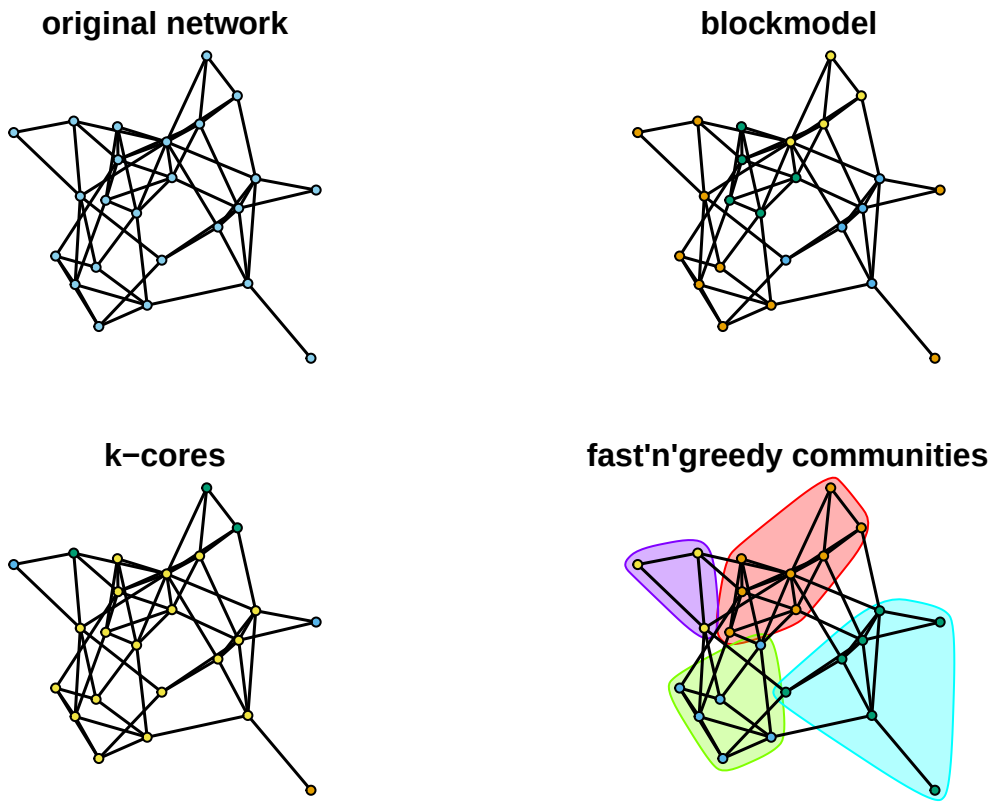
```
# Visualisatie:
plot(communities, class_graph,
     edge.color = "black",
     edge.width = 1.5,
     edge.arrow.size = 0.25,
     vertex.size = 6,
     vertex.label = "",
     layout=myLayout)
```



Laat ons nu een finale plot maken van alles samen:

```
par(mfrow=c(2,2), mar = c(2, 2, 1, 1))

# origineel netwerk
plot(class_graph,
```

```
     edge.color = "black",
     edge.width = 1.5,
     edge.arrow.size = 0.25,
     vertex.size = 6,
     vertex.label = "",
     vertex.color="skyblue",
     layout=myLayout,
     main="original network")

# blockmodel
plot(class_graph,
     edge.color = "black",
     edge.width = 1.5,
     edge.arrow.size = 0.25,
     vertex.size = 6,
     vertex.label = "",
     vertex.color= block_members,
     layout=myLayout,
     main="blockmodel")

# k-cores
plot(class_graph,
     edge.color = "black",
     edge.width = 1.5,
     edge.arrow.size = 0.25,
     vertex.size = 6,
     vertex.label = "",
     vertex.color= cores,
     layout=myLayout,
     main="k-cores")

# fast-greedy communities
plot(communities, class_graph, edge.color = "black",
     edge.width = 1.5,
     edge.arrow.size = 0.25,
     vertex.size = 6,
     vertex.label = "",
     layout=myLayout,
     main="fast'n'greedy communities")
```




```
par(mfrow=c(1,1))
```

# 7. Netwerk niveau (network level properties)

Op het netwerk niveau analyseren we globale eigenschappen van het gehele netwerk in plaats van individuele nodesn of kleine subgroepen. Het bestuderen van netwerk niveau kenmerken is essentieel om het gedrag en de functionele mogelijkheden van het netwerk als geheel te begrijpen.

## 7.1. Grootte van het netwerk

De grootte van het netwerk is simpelweg gelijk aan het aantal nodes in het netwerk:

```
# Gebruik het graph object
length(class_graph)
```

```
## [1] 24
```

```
# Print ook nog even het gemiddelde kortste pad van de 'main component'
igraph::mean_distance(main_component) # of via average.path.length()
```

```
## [1] 2.188406
```

```r
# Print ook nog even de diameter de 'main component'
igraph::diameter(main_component) # of via average.path.length()
```

```
## [1] 5
```

## 7.2. Density

De densiteit van een netwerk is een maat die aangeeft hoe dicht het netwerk is verbonden. Concreet is het de verhouding tussen het aantal aanwezige verbindingen (edges) en het maximaal mogelijke aantal verbindingen tussen alle nodes in het netwerk. Een hogere dichtheid wijst op een netwerk waarin relatief veel knopen direct met elkaar verbonden zijn, terwijl een lagere dichtheid duidt op een meer 'sparse' structuur met minder directe verbindingen. In sociale netwerken kan de dichtheid bijvoorbeeld inzicht geven in de mate van onderlinge verbondenheid binnen een groep:

```r
# Density van het netwerk (let op: in matrix formaat wat is sna function)
sna::gden(class_matrix, mode = 'digraph') # mode = 'digraph' -> directed; 'graph' voor undirected
```

```
## [1] 0.1394928
```

## 7.3. Degree distributie (zie node level, maar ook belangrijk op netwerk niveau)

## 7.4. Transitivity

Transitivity reflecteert de mate van clustering binnen een netwerk. Het geeft de kans aan dat 'adjacent'/naburige nodes van het netwerk verbonden zijn met elkaar. Dit weerspiegelt het idiom: een vriend van een vriend is ook mijn vriend (oftewel de aanwezigheid van gesloten driehoeken (triads) in het netwerk.) Een hoge transitiviteit betekent dat veel van deze driehoeken voorkomen, wat duidt op sterke lokale cohesie binnen het netwerk. Dit concept is belangrijk omdat het inzicht verschaft in de mate van groepsvorming en de potentie voor informatie- of gedragsverspreiding binnen sociale netwerken. Belangrijk is dat we dit berekenen op basis van de 'main component' van het netwerk, wat in dit geval het volledige netwerk omvat:

```r
# Transitivity van het netwerk (let op: gebruik hier igraph object wat is igraph functie)
igraph::transitivity(main_component)
```

```
## [1] 0.3345725
```

**Einde van deze notebook**