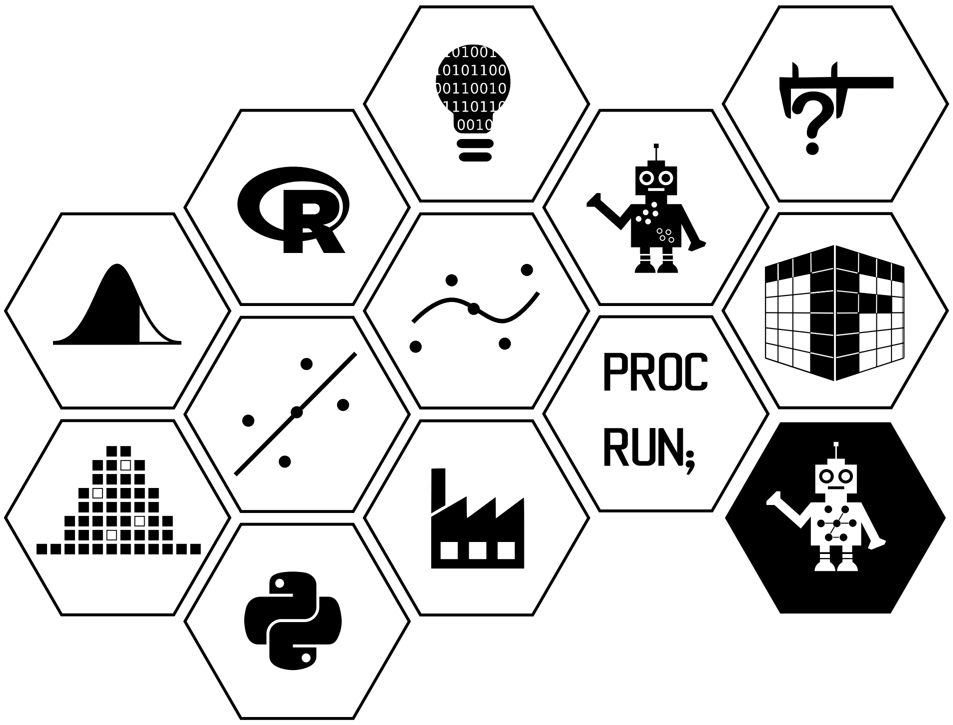
Data Mining and Machine Learning II

Charis Chanialidis & Nema Dean
Lightly edited by Andrew Elliott

Academic Year 2021-22

Week 1:

Network Analysis Part I



Statistical Network Analysis

A totally different kind of statistics!

The modern world, more than ever before, is about making connections. Buzz words like “networking” are commonplace in most workplaces. The “six degrees of separation” theory states that everyone and everything is six or fewer steps away, by way of introduction, from any other person in the world. The name for this theory is often used as a synonym for the “small world” phenomenon. For those of you old enough to remember the actor Kevin Bacon, there’s even a game version of this called “Six degrees of Kevin Bacon”, where people try to link any named actor to Kevin Bacon using no more than six connections. Obvious examples of online social media that exploit or allow interpersonal connections include Facebook and LinkedIn. In 2011, Facebook published a report that said the average distance between pairs of users was 4.74 with 99.91% of users connected.

The idea of connections and networks are important where one is interested in studying the connectivity of sets of people, organisations or any kind of object. Often when the number of items being studied is small, one can easily draw a picture of a graph representing the connections and do an analysis by eye. But if the number of objects is large or there are other pieces of information that we wish to explore related to the objects or connections, we need to use network analysis. Network analysis is a fairly recent area of research in statistics, combining ideas from graph theory in mathematics, computer science and research from other fields like anthropology, sociology and psychology. The beginnings of social network analysis is usually credited to Jacob Moreno in the 1930s. As a relatively young field, there are still lots of active researchers in the area as well as dedicated journals, like *Social Networks*.

Network analysis, often referred to as statistical network analysis or social network analysis, is a set of methods that help explore and analyse network data, be those networks of people, networks of countries, networks of genes or any other kind. The main goals are

- visualization
- summarizing network and network-related variable attributes
- statistical modelling of networks

We will explore aspects of each of these in this handout but obviously with an extremely wide variety of options, we cannot cover everything or always go into the detail we would like. This handout is based on personal notes, as well as material from the internet and textbooks. A good set of textbooks to get started with network analysis are given in the following supplementary material.

What is Social Network Analysis?
Simple defintion: Methodical analysis of social network
The analytical study of (social) relations amo set of actors
GLASGOW

Introduction to Social Network Analysis

https://youtu.be/TnHS62UgDVg

Duration: 3m09s

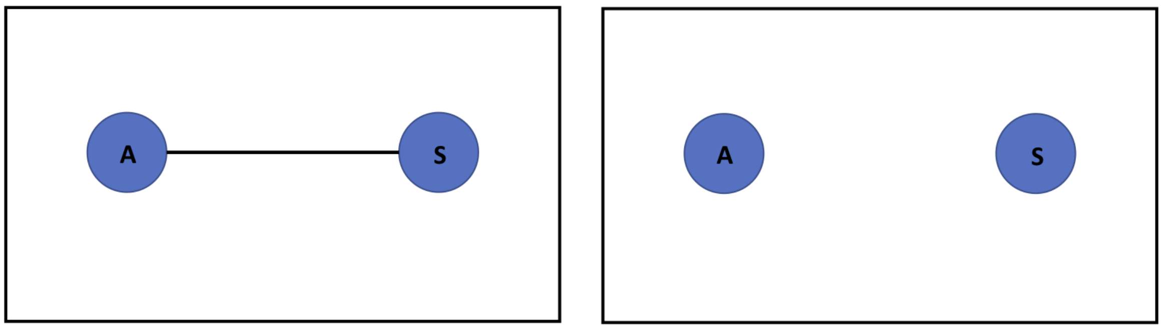
Supplementary material:

Good starter textbooks:

- Statistical Analysis of Network Data by Eric D. Kolaczyk (Springer), and its companion volume
- Statistical Analysis of Network Data with R by Eric D. Kolaczyk and Gábor Csárdi (Springer)
- A User’s Guide to Network Analysis in R by Douglas A. Luke (Springer)
- Networks: An Introduction by Mark Newman (Oxford University Press)

There are also numerous online resources including Robert A. Hanneman and Mark Riddle’s online introduction to social network methods at http://www.faculty.ucr.edu/~hanneman/nettext/.

We choose to use the R statistical software language to implement the topics introduced in the course because of the flexibility of the language, the excellent visualizations possible and the host of methods for which there are already implemented in R packages. However, R is not the only option available, nor is it always going to be the best one. Other stand-alone possibilities include Pajek or Gelphi as well as various high level programming languages e.g. Matlab, Python, Julia etc. For readers who use Python as their main language and wish to use these methods more widely,



(a) Friendship (b) No friendship

Figure 1: Two possibilities for (undirected) friendship

there are some excellent Python libraries for Network Analysis for example Networkx; iGraph or Graphtool.

The remainder of this handout will be divided as follows. The next section will introduce the terminology and notation used in this area and the Managing Data section will explain how to read in and import network data and change between different data formats in the R packages as needed. In terms of actually analyzing data, the Visualizing Network Data section will start off by discussing how to create meaningful plots of the graphs underlying the network data, the Network Summary Statistics section will introduce network characteristics of importance and how to calculate and interpret the statistics that measure these and the Network Models section will introduce the exponential random graph model as well as a couple of other popular models. The Stochastic Block Models section discusses a network clustering model known as the stochastic block model. Finally we conclude with a discussion which reviews some of the methods not covered and gives advice on finding out more about this subject.

Terminology, Notation and Simple Summaries

Terminology

The foundation of network analysis is the graph. Graphs are made up of two sets: a set of nodes/actors/vertices and a set of connections/links/ties/edges between those nodes. In mathematical terms we'd say that a graph G is a structure consisting of a set V of vertices and a set E of edges, with E being made up of pairs $\{u, v\}$ of distinct vertices $u, v \in V$ that define the edges. Note that this defines a binary network where either an edge between u and v is present, i.e. $\{u, v\} \in E$ or it is not, i.e. $\{u, v\} \notin E$. This can be extended to the idea of non-binary (or weighted) edges later on. But many networks of interest take a binary form and that is the type of network which has received the most attention in the literature.

The number of vertices/nodes, $|V|$, is known as the order of the graph G, while the number of edges, $|E|$ is known as the size of the graph.

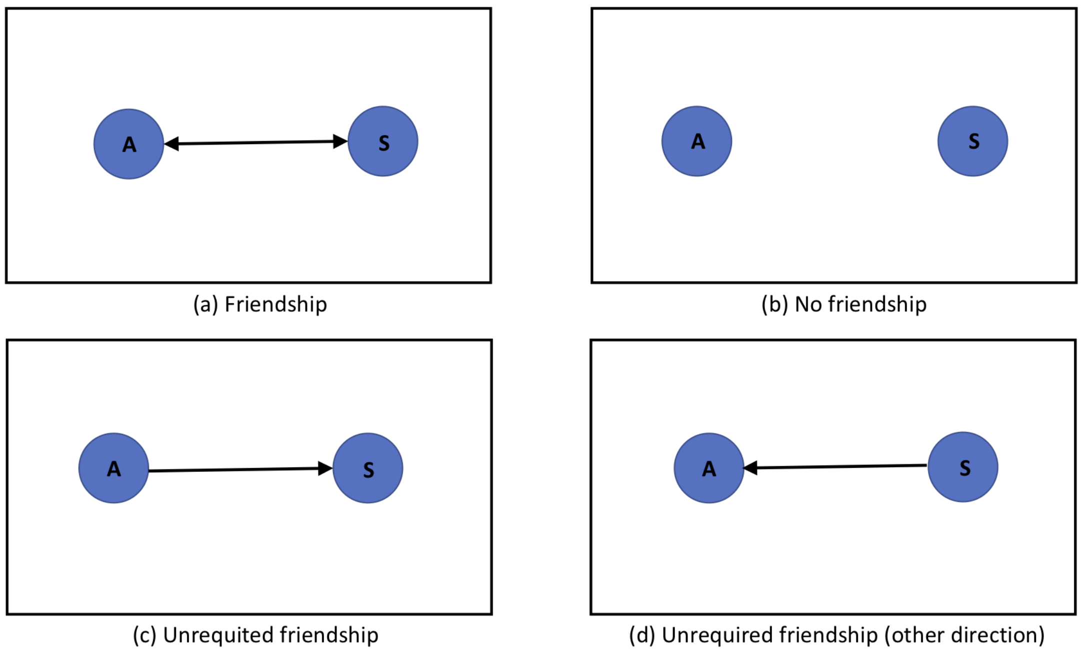
Network edges can be undirected or directed.

Example 1.

For example, if the question is 'are Ahmed and Sofia friends', the answer can be yes or no: as seen in Figure 1 'yes' indicates an edge exists between the nodes representing these two; 'no' indicates no edge between the two.

Alternatively, there can be two questions: 'does Ahmed consider Sofia a friend' and 'does Sofia consider Ahmed a friend', resulting in four possibilities seen in Figure 2: (a) both say yes so a directed edge goes from the node representing Ahmed to the one representing Sofia and vice versa; (b) both say no, so no edges exist between the two nodes; (c) Ahmed says 'yes' but Sofia says 'no' (ouch!) so an edge exists from Ahmed to Sofia but not from Sofia to Ahmed; or (d) Ahmed says 'no' but Sofia says 'yes' (again, ouch) so an edge exists from Sofia to Ahmed but not from Ahmed to Sofia.

In the undirected network case, if the edge $\{u, v\}$ exists in E then the edge {v,u} will also (or is assumed to, if we want



(a) Friendship (b) No friendship

(c) Unrequited friendship (d) Unrequired friendship (other direction)

Figure 2: Four possibilities for (directed) friendship

to avoid duplication). Whereas in the directed edge case, we have four possibilities for each pair of nodes u and v: both $\{u, v\}$ and $\{v, u\}$ are in E, only $\{u, v\}$ but not $\{v, u\}$ are in E, only $\{v, u\}$ but not $\{u, v\}$ are in E or neither $\{u, v\}$ and $\{v, u\}$ are in E.

A pair of nodes (whether connected or not) is often referred to as a **dyad**.

Notation

Network data can be represented in a number of different ways in practice. We can give a list of paired node names that represent edges which define the network. Or we can give a list of the node names and a square binary matrix (often known as a sociomatrix or adjacency matrix). The matrix has the same number of rows as columns and each row and column represents a node. An entry in the matrix at the ith row and jth column represents the presence with a 1 or absence with a 0 of an edge from the ith to jth node (if directed or between them if undirected).

Example 2.

So the Ahmed/Sofia undirected friendship question would be represented either by

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

if they're friends or

$$X = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

if not. For the directed friendship questions, the state of their friendship would be represented by

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

if both says the other is a friend, or

$$X = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

if neither says the other is a friend, or

$$X = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$$

if Ahmed says Sofia is a friend but not the reverse (assuming the first node is Ahmed and the second Sofia) or

$$X = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}$$

if Sofia says Ahmed is a friend but not the reverse.

Note that sociomatrices representing *undirected* edges will always be symmetric matrices, because $X_{ij} = X_{ji}$. Note also, that the diagonal entries are all zero, the convention being that a node cannot have an edge with itself. (A tie or edge of a node with itself would be called a loop.)

Graph representation
List of paired node names representing edges defining the graph
Sociomatrix/adjacency matrix: list of the nod names and a square bina matrix
GLASGOW

Network representations

https://youtu.be/PuEc4Od7vOU

Duration: 8m14s

Simple Summaries

If our graph/network information is in a sociomatrix form then we can easily calculate some simple descriptive statistics using some matrix manipulation.

Graph size The size of a graph is equal to the number of ties that are observed in it. For a directed graph, this can be found by summing the entries of the sociomatrix (directed size=$\sum_{i=1}^{k}\sum_{j=1}^{k} X_{ij}$), for an undirected graph it is given by the summation of the sociomatrix entries divided by 2 (undirected size = $\sum_{i=1}^{k}\sum_{j=1}^{k} X_{ij}/2$). We could code this in R as:

```
# For a directed graph
size.dir.X <- sum(X)
# For an undirected graph
size.undir.X<-sum(X)/2
```

Graph density The density of a graph is related to how many edges are observed versus how many edges are possible given the number of nodes in the graph. If we are in a directed graph with every possible pair of the k nodes having an edge, we would have $k \times (k-1)$ edges. In an undirected graph this will be halved, i.e. $k \times (k-1)/2$. So the closer our observed number of edges is to this theoretical maximum, the more dense (or connected) our network is. So the density of a graph is given by the equation:

$$\text{graph density} = \frac{\text{number of edges observed in the graph}}{\text{maximum number of edges possible in the graph}}.$$

To find the number of edges in a directed graph, we simply add together all the entries in the sociomatrix so our equation becomes:

$$\text{density of directed graph } X = \frac{\sum_{i=1}^{k}\sum_{j=1}^{k} X_{ij}}{k \times (k-1)}.$$

To find the number of edges in an undirected graph, we simply add together all the entries in the sociomatrix and divide by 2 so our equation becomes:

$$\text{density of undirected graph } X = \frac{\sum_{i=1}^{k}\sum_{j=1}^{k} X_{ij}/2}{k \times (k-1)/2} = \frac{\sum_{i=1}^{k}\sum_{j=1}^{k} X_{ij}}{k \times (k-1)}.$$

We could code this in R as:

```
dens.X <- sum(X)/(nrow(X)*(nrow(X)-1))
```

Notice that both the directed and undirected graphs use the same calculation for density (as the 2's cancel out).

Reciprocity If we are in a directed graph and node u has an edge with v and also, node v has an edge with u, then we say the tie is reciprocated. Often of interest in directed graphs is the proportion of times a tie is reciprocated. We can calculate this by counting the number of times $X_{ij} = X_{ji} = 1$ and dividing by the number of pairs of nodes where we have at least one tie.

$$\text{Reciprocity of X} = \frac{\sum_{i \neq j, i<j} 1[X_{ij} = X_{ji} = 1]}{\sum_{i \neq j, i<j} 1[X_{ij} + X_{ji} > 0]},$$

where $1[\ ]$ is an indicator function which gives the value 1 if the statement in the brackets is true and 0 otherwise. Using the transpose of the matrix (where X_{ij} becomes X_{ji}), we can code this simply in R as:

```
X.Xt<-X*t(X) ##this will give a matrix with 1's for reciprocal ties
no.recip.ties<-sum(X.Xt==1)/2
no.non.recip.ties<-sum(X-X.Xt)
prop.recip.X<-no.recip.ties/(no.non.recip.ties+no.recip.ties)
```

Directed graphs with all ties reciprocated will have reciprocity equal to 1, while a purely unidirectional graph will have reciprocity equal to 0. Most directed graphs have a value that falls between 0 and 1. Other more complex summaries will be discussed in the section on Summary Statistics.

Node degree For a directed graph, we define in-degree as the number of edges leading into a node and out-degree as the number of edges originating from a node. In terms of the friendship example, in-degree would be how popular a person was (how many people said they were friends with that person) and out-degree would be how outgoing that person was (how many people that person said they were friends with). For node i the in-degree would be calculated as:

$$\text{in-degree of node } i = \sum_{j=1}^{k} X_{ji}, \tag{1}$$

i.e. the sum of the i^{th} column of X and out-degree would be:

$$\text{out-degree of node } i = \sum_{j=1}^{k} X_{ij}, \tag{2}$$

i.e. the sum of the i^{th} row of X. This can be simply calculated in R for all nodes as:

```
in.degree<-colSums(X)
out.degree<-rowSums(X)
```

Note there are simple commands in the various SNA packages that perform these functions, so you won't have to do them from scratch but it's good to have an idea what they are doing.

Managing Network Data

Reading in different forms of network data

Sociomatrix by hand One way of entering network information by hand is to enter the sociomatrix into R using the `matrix` command. The reciprocal or undirected Ahmed/Sofia graph would be entered via the following:

```
sociomatrix<-matrix(c(0,1,1,0),2,2,byrow=T)
rownames(sociomatrix)<-colnames(sociomatrix)<-c("Ahmed","Sofia")
```

Suppose we wanted to enter a more complex graph like the following:

```
sociomatrix<-matrix(c(0,1,0,0,1,0,0,0,1,1,0,1,0,1,1,0),4,4,T)
rownames(sociomatrix)<-colnames(sociomatrix)<-c("Ahmed","Sofia","Berthold","Carlo")
```

Task 1.

Take a look at what this command produces and try to describe the friendship structure that you see in words. How many edges does the graph have? How dense is it? What proportion of friendship ties are reciprocated? What is the distribution of in-degree and out-degree of the nodes? Who is the most popular person in the network? Who is the friendliest?

We could also read in data in the sociomatrix form from a matrix recorded in an external file (say a .csv or .txt file) use the `read.table` or `read.csv` command. We will go through an example of this later.

One disadvantage of sociomatrices is that as the number of nodes increases, the number of entries to be stored increases dramatically. In a lot of network examples, many of these entries will be zero, i.e. the matrix is sparse, so a more efficient way of storing the information may be as a list of edges instead. Each entry/row in such a list will be a pair of node identifiers with an implied edge going from the first node entry to the second one.

List of edges by hand We can create a network using a list of edges in the following way. For our friendship example we had a total of 7 edges (2 reciprocated, 3 not) which could be read in as follows:

```
friend.edges<- rbind(c("Ahmed","Sofia"),
                     c("Sofia","Ahmed"),
                     c("Berthold","Ahmed"),
                     c("Berthold","Sofia"),
                     c("Berthold","Carlo"),
                     c("Carlo","Sofia"),
                     c("Carlo","Berthold"))
friend.edges<-data.frame(friend.edges)
```

We will talk in the next section about changing this format into different types of network objects.

Changing to network format for `statnet`

If we have a sociomatrix or edge list, we may need to transform it into a particular class of `R` object in order to be able to use a particular library's commands to analyse the network. For the `statnet` library, we want to create a `network` object using the `network` command. For example using this command to format a sociomatrix, we set the `matrix.type` argument to `"adjacency"`:

```
library(statnet)
```

```
sociomatrix<-matrix(c(0,1,0,0,1,0,0,0,1,1,0,1,0,1,1,0),4,4,T)
rownames(sociomatrix)<-colnames(sociomatrix)<-c("Ahmed","Sofia","Berthold","Carlo")
net1<-network(sociomatrix,matrix.type="adjacency")
```

Note that if you do not give the matrix row and column names, the default labeling for nodes will just be numbers.

Task 2.

Take a look at the `summary` of the object created with the previous code. This can be a useful check to see if the network object was successfully created.

If, instead of a sociomatrix, we have an edgelist, we set the `matrix.type` argument in the `network` command to `"edgelist"` instead. That is,

```
net1.1<-network(friend.edges, matrix.type="edgelist")
```

Task 3.

Check to see that this will give the same network as before.

Note that if we have left out node names earlier in the process, we can add them to the network object using the `network.vertex.names`

```
network.vertex.names(net1)<-c("Ahmed","Sofia","Berthold","Carlo")
```

If you need to go in the other direction you can use the `as.sociomatrix` command to force a network object into a matrix format. Similarly you can use the `as.matrix` command to extract a list of edges by using the following code:

```
as.matrix(net1, matrix.type="edgelist")
```

Adding other information to the network in `statnet`

Other information for a network could include node covariates, edge covariates and other information about the network itself (e.g. whether it's directed, if loops are allowed, etc.).

Adding node characteristics to a network object In `statnet` we can use the `set.vertex.attribute` command to attached node characteristics to the network object. For example, if we wanted to add gender information to our friendship network:

```
network::set.vertex.attribute(net1, "gender",c("M","F","M","M"))
```

If we take a look at the summary of the network object now, we see that it includes a tabulation of our categorical vertex attribute gender (if it were a continuous attribute it would have given a 5-number summary of it instead, i.e. min, median, max, 1st and 3rd quartiles)

```
summary(net1)
```

```
## Network attributes:
##   vertices = 4
##   directed = TRUE
##   hyper = FALSE
##   loops = FALSE
##   multiple = FALSE
##   bipartite = FALSE
##  total edges = 7
##    missing edges = 0
##    non-missing edges = 7
##  density = 0.5833333
##
## Vertex attributes:
##
##  gender:
##    character valued attribute
##    attribute summary:
## F M
## 1 3
##   vertex.names:
##    character valued attribute
##    4 valid vertex names
##
## No edge attributes
##
## Network adjacency matrix:
##          Ahmed Sofia Berthold Carlo
## Ahmed        0     1        0     0
## Sofia        1     0        0     0
## Berthold     1     1        0     1
## Carlo        0     1        1     0
```

To check what vertex attributes are available with a network object we can use the `list.vertex.attributes` command which will list the names of all covariates. To extract covariate values we use the `get.vertex.attribute` as in the following example:

```
network::get.vertex.attribute(net1,"gender")
```

```
## [1] "M" "F" "M" "M"
```

```
network::get.vertex.attribute(net1,"vertex.names")
```

```
## [1] "Ahmed"    "Sofia"    "Berthold" "Carlo"
```

An equivalent way of extracting this information is using the %v% command, e.g. net1 %v% "gender". If you look at the `summary` of a network object with node covariates, it will include tabulation summaries of categorical covariates and summary statistics for continuous covariates. To remove a node covariate, we use the `delete.vertex.attribute` command.

Adding edge characteristics to a network object Adding edge covariates follows the same form as for nodes (except edge covariates will be in matrix form while node covariates will be vectors), using commands `set.edge.value` to add a covariate (in matrix form), `get.edge.attribute` or `get.edge.value` to extract the covariate values in list form and `list.edge.attributes` to get a list of the edge covariates available in the object. For example if we had a weight or strength of association measure between pairs of people, we could use that as an edge covariate:

```
w.mat<-matrix(c(0,3,0,0,2,0,0,0,1,1,0,3,0,1.5,4,0),4,4,T)
set.edge.value(net1,"weight",w.mat)
```

If we want to extract the covariate in matrix form we can use the `as.sociomatrix` command in the following way:

```
as.sociomatrix(net1,"weight")
```

```
##          Ahmed Sofia Berthold Carlo
## Ahmed        0   3.0        0     0
## Sofia        2   0.0        0     0
## Berthold     1   1.0        0     3
## Carlo        0   1.5        4     0
```

To remove an edge covariate, use the `delete.edge.attribute` command. For further information on these methods, look at the help file for `attribute.methods`.

Changing to network format for `igraph`

First make sure you detach the `statnet` packages before loading the `igraph` library (this is to avoid masking functions with the same names in both).

```
detach("package:statnet", unload=TRUE)
detach("package:tsna",unload = TRUE)
detach("package:sna", unload=TRUE)
detach("package:ergm.count", unload=TRUE)
detach("package:tergm", unload=TRUE)
detach("package:ergm", unload=TRUE)
detach("package:networkDynamic", unload=TRUE)
detach("package:network", unload=TRUE)
```

```
## Warning: 'network' namespace cannot be unloaded:
##   namespace 'network' is imported by 'intergraph' so cannot be unloaded
```

```
library(igraph)
```

Similar to `statnet`, you can either create a network using a sociomatrix or an edge list. Using the `graph_from_adjacency_matrix` command we can read in the same friendship matrix as before:

```
inet1<-graph_from_adjacency_matrix(sociomatrix)
```

This will create an `igraph` network object. Looking at a `summary` of this object will be a little more cryptic than before:

```
summary(inet1)
```

```
## IGRAPH 08cee1a DN-- 4 7 --
## + attr: name (v/c)
```

Here 'D' denotes directed graph, 'N' denotes named nodes. The first number is the number of nodes and the second, the number of ties.

We can use the command `graph_from_edgelist` to produce a similar object from an edge list instead of a sociomatrix.

```
inet1.1<-graph_from_edgelist(as.matrix(friend.edges),directed = T)
```

Adding other information to the network in `igraph`

Adding node characteristics to an igraph object The `V` command allows us to add node/vertex covariates to an igraph object or view covariates as well, in the following way:

```
#Adding gender
V(inet1)$gender <- c("M","F","M","M")
#Looking at the gender covariate
summary(V(inet1)$gender)
```

```
##    Length     Class      Mode
##         4 character character
```

Gender will now appear as part of the `summary` output.

Adding edge characteristics to an igraph object

The `E` command allows us to add edge covariates to an igraph object or view covariates as well, in the following way:

```
#Adding a weight for strength of friendship
E(inet1)$strength <- c(3,3,1,2,2,1,4)
#Looking at the strength covariate
summary(E(inet1)$strength)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   1.500   2.000   2.286   3.000   4.000
```

If we look at the summary information now,

```
summary(inet1)
```

```
## IGRAPH 08cee1a DN-- 4 7 --
## + attr: name (v/c), gender (v/c), strength (e/n)
```

we can see the covariates listed with 'v' or 'e' depending on whether it is a vertex or edge covariate and 'n' or 'c' depending on whether the covariate is numeric or categorical.

Converting between `statnet` and `igraph`

The library `intergraph` allows us to convert from a `statnet` network object to an `igraph` object and vice versa. The command to force into network object is `asNetwork` and the command to force into igraph object is `asIgraph`.

```
library(intergraph)
inet1.2<-asIgraph(net1)
net1.2<-asNetwork(inet1)
```

Importing from other file types

The `statnet` package allows reading in Pajek format file (.net or .paj) using the `read.paj` function. `igraph` allows something similar for Pajek files as well as GraphML and UCINet DL files using the `read_graph` command.

Extracting sub-graphs and isolates

Extracting sub-graphs based on node characteristics

Suppose we wanted only to look at the subgraph of male nodes in the friendship network example, we can use the command `get.inducedSubgraph` to do so in the following way:

```
##                 Installed ReposVer Built
## ergm            "4.0.1"   "4.1.2"  "4.0.4"
## networkDynamic "0.11.0"  "0.11.1" "4.0.2"
## tergm           "4.0.1"   "4.0.2"  "4.0.2"
## tsna            "0.3.3"   "0.3.5"  "4.0.4"
```

```
library(statnet)
```

```
netM<-get.inducedSubgraph(net1,which(net1 %v% "gender" == "M"))
```

Similarly if we had a numeric covariate we could select a subgraph based on cut-offs or ranges of values for this covariate in a similar fashion. Instead of the `get.inducedSubgraph` command we could also use the `%s%` command.

```
netnew <- net1 %s% which(numeric.covariate>1)
```

Extracting sub-graphs based on edge characteristics

If we are only interested in those set of edges with a certain value of attribute, we can look at summaries of the number of edges with certain values in the following way:

```
table(net1 %e% "weight")
```

```
##
##   1 1.5   2   3   4
##   2   1   1   2   1
```

```
#If we were only interested in edges with weight greater than 2, say
#First we extract a matrix version of the edge covariate
w.val<-as.sociomatrix(net1,"weight")
#Replace the elements below the threshold with 0
w.val[w.val<2]<-0
#Create a new sociomatrix with ties corresponding to the new zeros filtered out
net.filt<-as.network(w.val,directed=TRUE,matrix.type="a",ignore.eval=FALSE,
names.eval="weight")
```

Removing isolates Isolates are nodes in a graph that are disconnected from all other nodes. There are no edges connecting them to others, i.e. in-degree and out-degree are both 0.

Let's create a new friendship sociomatrix with a newcomer who hasn't had a chance to make friends:

```
new.soc<-rbind(cbind(sociomatrix,rep(0,4)),rep(0,5))
rownames(new.soc)<-colnames(new.soc)<-c("Ahmed","Sofia","Berthold","Carlo","Sean")
net.new<-network(new.soc)
```

The command `isolates` will return the indices of those nodes with degree 0. We can use the `delete.vertices` command to remove these but it is a good idea to create a copy of the network object to do this on so that you still have the original to return to.

```
net.copy <- net.new
delete.vertices(net.copy,isolates(net.copy))
```

Changing a directed graph to undirected The command to change from a directed to undirected graph is `symmetrize` (appropriately enough if you recall the sociomatrix of an undirected graph is symmetric).

```
net.symm.weak<-symmetrize(net1,rule="weak")
net.symm.strong<-symmetrize(net1,rule="strong")
```

The argument `rule` being set to `"weak"` means an undirected edge will be defined between any two nodes that have at least one directed edge connecting them. In comparison, setting this to `"strong"` would require that only reciprocated edges will be allowed in the new network. Different rules suit different situations.

This command will produce an unlabeled symmetric matrix that will need to be converted into a network or igraph object again.

Task 4.

Compare the two different symmetrized matrices for differences.

Doctor network analysis example

Task 5.

Read in the adjacency matrix `ckm_network.dat` (use the `read.table` command). Check that there are 246 nodes. This is from one of the classic studies of network analysis and the diffusion of innovations which concerned the spread of a new antibiotic among the doctors in a group of four small towns in the Midwest in the 1950s. The ties indicate pairs of doctors that were socially linked (undirected tie). Explore the data using the skills from the previous sections and record your observations in the space provided. What do the degree distributions of nodes look like? Are there any isolates?...

Visualizing Network Data

One of the most powerful ways to explore network data is visually. For reasonable size graphs, this should be the first thing you look at (unfortunately for graphs of large order, say tens of thousands of nodes or more, this tends to result in an uninformative blob of ink). Because there are no natural co-ordinates for nodes in most circumstances, plotting the configuration of nodes can take any number of forms. Some types of plotting can make evaluating certain graph characteristics easier than others so the type of plot you use should be chosen with that goal in mind. We'll discuss a couple of different graph plotting methods but there are many more out there if you look.

The standard `plot` command in `statnet` allows for a 2-d graph visualization with options for adding text labels to the nodes, different colours according to node categories and has a number of different algorithms to decide on the node configuration.

Example 3.

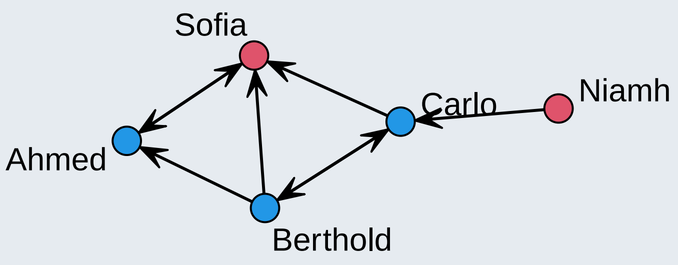
Plotting some friendship data, we can add the individual's name, colour the node differently according to gender and manipulate the graph options to change the size of the lines, arrowheads and node circles.

Here we add a new actor "Niamh" (who unlike Sean from the previous example, actually made a friend!).

```
sociomatrix<-matrix(c(0,1,0,0,0,1,0,0,0,0,1,1,0,1,0,0,1,1,0,0,0,0,0,1,0),5,5,T)
rownames(sociomatrix)<-colnames(sociomatrix)<-c("Ahmed","Sofia","Berthold","Carlo",
"Niamh")
```

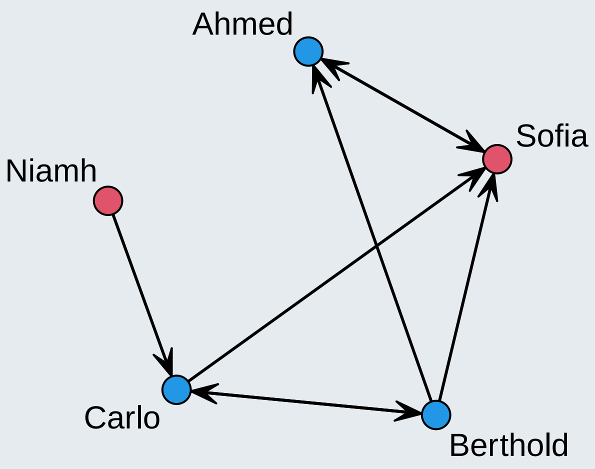
```
library(statnet)
```

```
net3<-network(sociomatrix,matrix.type="adjacency")
plot(net3,vertex.col=c(4,2,4,4,2),vertex.cex=3,label=network.vertex.names(net3),
arrowhead.cex=3,edge.lwd=2)
```



This command uses the default plotting method which is `fruchtermanreingold`, a variant of Fruchterman and Reingold's force-directed placement algorithm. This algorithm tries to pull together groups of nodes that are closely linked and push apart ones that are not. It works by iteratively adjusting the configuration of the network until the optimum of some measure of overall network energy has been found. This can be changed by changing the `mode` argument.

```
plot(net3,vertex.col=c(4,2,4,4,2),vertex.cex=3,label=network.vertex.names(net3),
arrowhead.cex=3,edge.lwd=2,mode="circle")
```

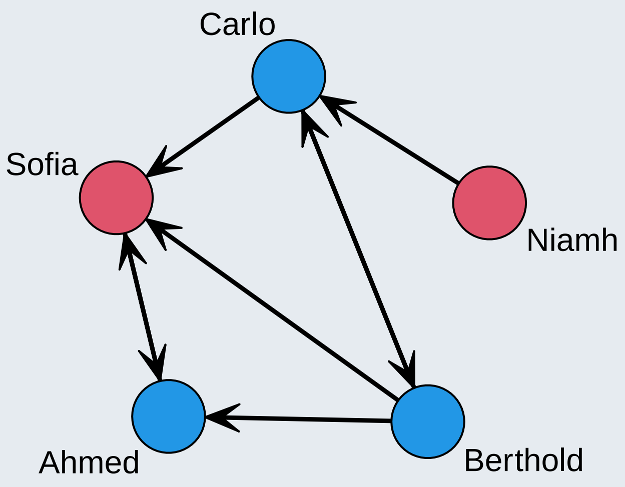


Note, if you run the same command again, you'll get a slightly different plot.

It's a good idea to run a couple of different visualizations to see what they show but **remember** that *line length is arbitrary* in these plots, so a pair of nodes connected by a **short** line are **no more connected** than if it had been a **long** line.

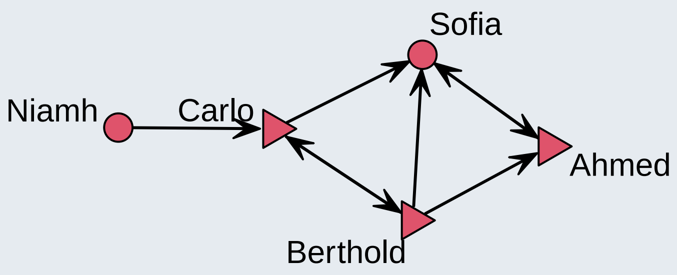
In addition to the basic `plot` command in `statnet`, there is the slightly more flexible command `gplot` which also has the circle and Fruchterman and Reingold methods but has additional options like `random`, eigen and spring. This is very similar to the plot command and an example is given below.

```
gplot(net3,vertex.col=c(4,2,4,4,2),vertex.cex=3,label=network.vertex.names(net3),
arrowhead.cex=1.5,edge.lwd=2, mode="spring")
```



Rather that indicating grouping of nodes by colour, we could also use different node shapes to distinguish them as in the following code and resulting plot:

```
plot(net3,vertex.sides=c(3,50,3,3,50),vertex.cex=3,label=network.vertex.names(net3),
arrowhead.cex=3,edge.lwd=2)
```



Note that 50 sides (vertex.sides=50) is the default, which produces a circle. Of course, you could use both colour and shape, with one attached to one categorical covariate and the other attached to another.

The igraph library also has a function plot.graph which works in a similar fashion. Greater control over manual adjustment of network graph co-ordinates is possible, see Chapter 4 in D. Luke's *A User's Guide to Network Analysis in R* for details. He also discusses how to make node colours partially transparent to help with overlap.

Interactive graphics

Gelphi is probably the best application for the creation of interactive network visualizations but there is some limited interactive graphing functionality in the igraph package using the tkplot command, which allows manual manipulation of graphs (and storing of the final manipulated co-ordinate system).

```
library(intergraph)
inet3<-asIgraph(net3)
library(igraph)
coord<-tkplot(inet3, vertex.size=3, vertex.label=c(4,2,4,4,2))
MCoords <- tkplot.getcoords(coord)
plot(inet3, layout=MCoords, vertex.size=5, vertex.label=NA, vertex.color="lightblue")
```

There is also potential functionality for web-publishing interactive network graphics (see Chapter 6 of D. Luke's *A User's Guide to Network Analysis in R* or Chapter 3 of *Statistical Analysis of Network Data with R* by Eric D. Kolaczyk and Gábor Csárdi.

ggplot2 style network plots

If you prefer to use the ggplot framework for visualizing data then the library GGally has a command ggnet that allows for plotting network objects in this way.

Task 6.

We are going to look at the doctor data network. Read in the adjacency matrix `ckm_network.dat` (use the `read.table` command). The ties indicate pairs of doctors that were socially linked (undirected tie). Take a look at some plots of this data. What interesting elements jump out at you? Which type of plot did you find most helpful?

Start by loading in the data and transforming it into a network object.

```
doc.data<-read.table("ckm_network.dat")
```

```
library(statnet)
```

```
set.seed(123)
doc.net<-network(as.matrix(doc.data), directed = F,matrix.type="adjacency")
```

Supplementary material:

You can do a lot of really clever things with plotting networks beyond the basics discussed here.

Some really nice network visualisation examples:

1. Porfiry: Visualisation of the Mueller Investigation
2. Non-standard graph representation of patients moving between different states of health
3. Dynamic visualisation of relationships between ancient philosophers

Answers to tasks

Answer to Task 1. First let's look at the adjacency matrix:

```
sociomatrix<-matrix(c(0,1,0,0,1,0,0,0,1,1,0,1,0,1,1,0),4,4,T)
rownames(sociomatrix)<-colnames(sociomatrix)<-c("Ahmed","Sofia","Berthold","Carlo")
sociomatrix
```

```
##          Ahmed Sofia Berthold Carlo
## Ahmed        0     1        0     0
## Sofia        1     0        0     0
## Berthold     1     1        0     1
## Carlo        0     1        1     0
```

We see zeros along the diagonal, as expected with some people's rows/columns having more 1's than others, e.g. Carlo seems more outgoing (two 1's in row) than Sofia (one 1 in row) but he seems to be less popular (only one 1 in column) than her (three 1's in column) too.

We now use our previous graph summaries calculation code to examine the overall structure of the graph:

```
X<-sociomatrix
#edges (this is a directed graph so we use the first option)
size.dir.X <- sum(X)
size.dir.X
```

```
## [1] 7
```

```
#density
dens.X <- sum(X)/(nrow(X)*(nrow(X)-1))
dens.X
```

```
## [1] 0.5833333
```

```
#reciprocity
X.Xt<-X*t(X) ##this will give a matrix with 1's for reciprocal ties
no.recip.ties<-sum(X.Xt==1)/2
no.non.recip.ties<-sum(X-X.Xt)
prop.recip.X<-no.recip.ties/(no.non.recip.ties+no.recip.ties)
prop.recip.X
```

```
## [1] 0.4
```

```
#in-degree
in.degree<-colSums(X)
in.degree
```

```
##    Ahmed    Sofia Berthold    Carlo
##        2        3        1        1
```

```
#out-degree
out.degree<-rowSums(X)
out.degree
```

```
##    Ahmed    Sofia Berthold    Carlo
##        1        1        3        2
```

So we can see there are 7 edges for the 4 friends which represents a density of 0.58 which is reasonably dense (over 50% of the possible edges). Only 40% of ties were reciprocated (this is for non-null ties, we're not counting the number of times neither person connected). Sofia is our most popular person with the largest in-degree while Berthold is our most out-going person with the largest out-degree.

Answer to Task 2.

```
library(statnet)
```

```
sociomatrix<-matrix(c(0,1,0,0,1,0,0,0,1,1,0,1,0,1,1,0),4,4,T)
rownames(sociomatrix)<-colnames(sociomatrix)<-c("Ahmed","Sofia","Berthold","Carlo")
net1<-network(sociomatrix,matrix.type="adjacency")
summary(net1)
```

```
## Network attributes:
##   vertices = 4
##   directed = TRUE
##   hyper = FALSE
##   loops = FALSE
##   multiple = FALSE
##   bipartite = FALSE
##  total edges = 7
##    missing edges = 0
##    non-missing edges = 7
##  density = 0.5833333
##
## Vertex attributes:
##   vertex.names:
##    character valued attribute
##    4 valid vertex names
##
## No edge attributes
##
## Network adjacency matrix:
##          Ahmed Sofia Berthold Carlo
## Ahmed        0     1        0     0
## Sofia        1     0        0     0
## Berthold     1     1        0     1
## Carlo        0     1        1     0
```

We can see the output matches the summaries we produced before (although it does not automatically give reciprocity or degree information).

Answer to Task 3. Let's take a look at the alternative way of specifying the adjacency matrix.

```
friend.edges<- rbind(c("Ahmed","Sofia"),
                     c("Sofia","Ahmed"),
                     c("Berthold","Ahmed"),
                     c("Berthold","Sofia"),
                     c("Berthold","Carlo"),
                     c("Carlo","Sofia"),
                     c("Carlo","Berthold"))
friend.edges<-data.frame(friend.edges)
net1.1<-network(friend.edges, matrix.type="edgelist")
summary(net1.1)
```

```
## Network attributes:
##   vertices = 4
##   directed = TRUE
##   hyper = FALSE
##   loops = FALSE
##   multiple = FALSE
##   bipartite = FALSE
##  total edges = 7
##    missing edges = 0
##    non-missing edges = 7
##  density = 0.5833333
##
## Vertex attributes:
##   vertex.names:
##    character valued attribute
##    4 valid vertex names
##
## No edge attributes
##
## Network adjacency matrix:
```

```
##          Ahmed Sofia Berthold Carlo
## Ahmed        0     1        0     0
## Sofia        1     0        0     0
## Berthold     1     1        0     1
## Carlo        0     1        1     0
```

This is identical to the previous output.

Answer to Task 4. Looking at the matrices themselves:

```
net.symm.weak<-symmetrize(net1,rule="weak")
net.symm.strong<-symmetrize(net1,rule="strong")
net.symm.weak
```

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

```
net.symm.strong
```

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

we can see that in the latter case (i.e. when `rule` is set to `"strong"`) the 1's that appear in the matrix reflect the reciprocated edges of the directed graph.

Answer to Task 5. First we need to read in the data and put it in an appropriate format.

```
doc.data<-read.table("ckm_network.dat")
library(statnet)
```

Next let's calculate some summaries by hand. (We can compare them later using the summary command). We do not need to calculate reciprocity as this is an undirected graph.

```
#Size
sum(doc.data)/2
```

```
## [1] 924
```

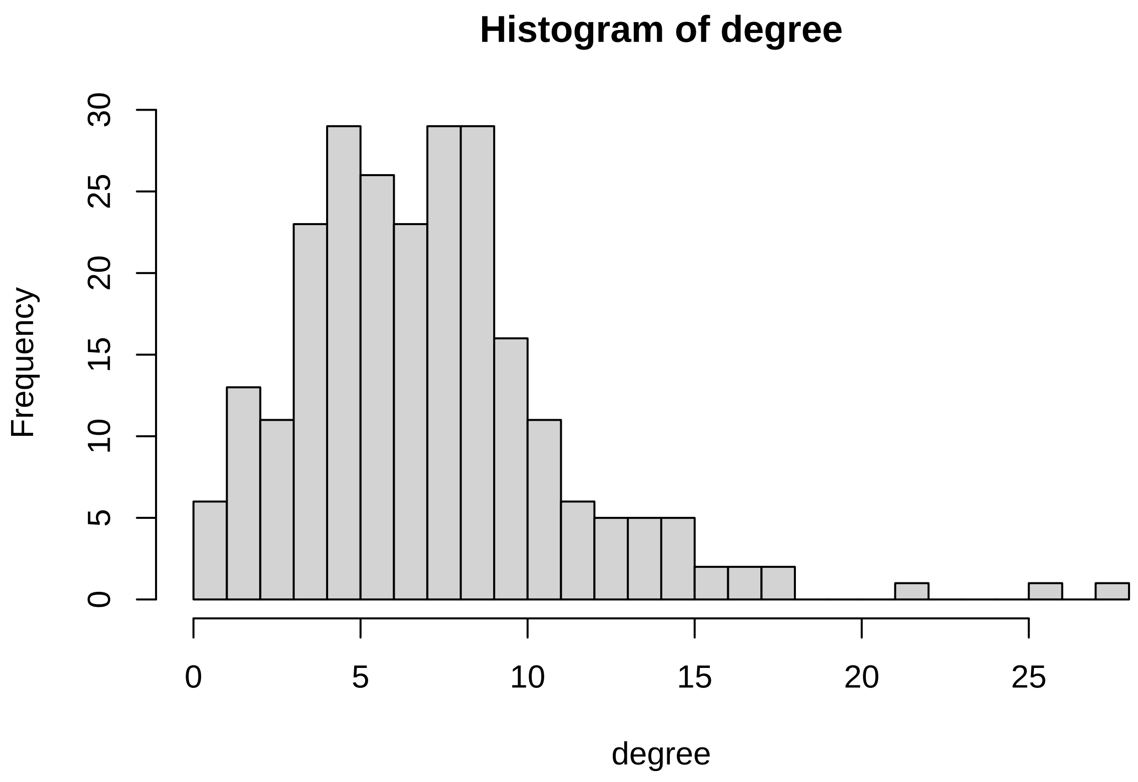
```
#Reciprocity not needed here (as symmetric matrix)

#Density
sum(doc.data)/(nrow(doc.data)*(nrow(doc.data)-1))
```

```
## [1] 0.03066202
```

```
#Node degree
degree<-rowSums(doc.data)
summary(degree)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.000   5.000   7.000   7.512   9.000  28.000
```

```
hist(degree,breaks = 20)
```



Note that in the `hist` function, we look at different numbers of boxes or splits summarising the data using the `breaks` argument. Here we set it to 20 but this can be changed and it's worth playing around with this to see how it affects the resulting plot.

We see that although it seems a big graph with 924 (undirected) edges, it actually has very few edges for the number of nodes (density is only 3%). The histogram of node degree tells us that there are a few very influential or popular doctors (the extreme values above 20) but most doctors have between 5 and 9 connections with an average number of connections of around 7/8. (Remember as there are 246 nodes, the maximum number of connections any one node can possibly have is 245. None of our doctors have anywhere near that amount.)

Next, let's turn this into a network object and check our previous work.

```
doc.net<-network(as.matrix(doc.data), directed = F,matrix.type="adjacency")
summary(doc.net,print.adj=F)
```

```
## Network attributes:
##   vertices = 246
##   directed = FALSE
##   hyper = FALSE
##   loops = FALSE
##   multiple = FALSE
##   bipartite = FALSE
##  total edges = 924
##    missing edges = 0
##    non-missing edges = 924
##  density = 0.03066202
##
## Vertex attributes:
##   vertex.names:
##    character valued attribute
##    246 valid vertex names
##
## No edge attributes
```

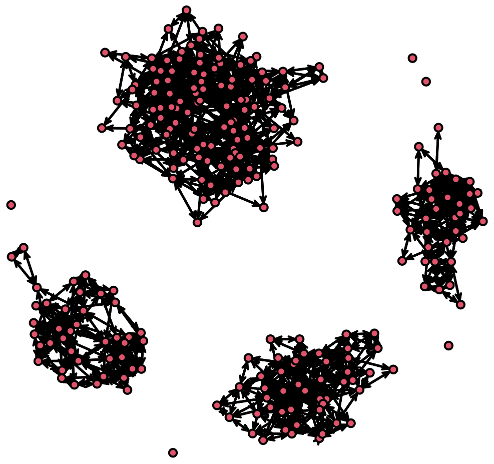
```
isolates(doc.net)
```

```
## [1] 154 165 195 201 203
```

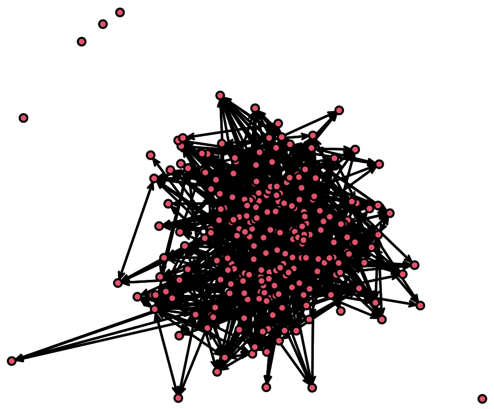
We see that there are 5 doctors with no connections at all.

Answer to Task 6. Let's take a look at a couple of different representations of the doctor data:

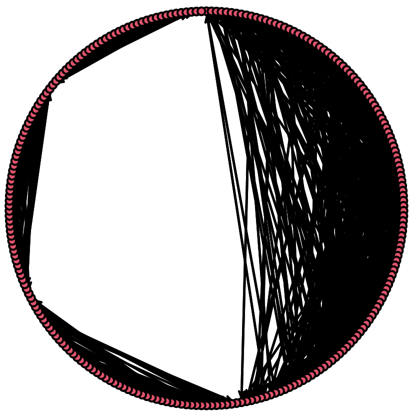
```
gplot(doc.net)
```



```
gplot(doc.net,mode="spring")
```



```
gplot(doc.net,mode="circle")
```



From the first and third plots, it is clear there is a group structure to this data with one large group and three small groups. While we can see the isolates in the second plot, the group structure is not as clear and in the third plot, the isolates are not clear. So in these terms, the first type of plot seems to be best.