Intro_to_SNA_R_Code.R

breannechryst

Thu Apr 7 14:42:15 2016

```r
## Introduction to SNA with R
## Breanne Chryst and
## CSSSI StatLab
## Feb 19, 2016

##################################################################################
## R Basics ##
##################################################################################
## R can be used as a calculator, it works as expected:
2+3
```

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

```r
exp(2)
```

```
## [1] 7.389056
```

```r
5^(2)
```

```
## [1] 25
```

```r
## Assigning a variable
x <- 5 # 5 has now been assigned to the variable x
x
```

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

```r
x^2
```

```
## [1] 25
```

```r
## Creating a vector:
y <- c(3,7,5,1,2,3,2,5,5) # "c()" concatenates, creating a vector

## Extracting values of a vector:
y[2]
```

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

```r
3:5 # the whole numbers from 3 to 5
```

```
## [1] 3 4 5
```

```r
y[3:5]
```

```
## [1] 5 1 2
```

```r
## "matrix()" creates a matrix from the values entered:
z <- matrix(y, nrow=3) # This is filled by column
z
```

```
##      [,1] [,2] [,3]
## [1,]    3    1    2
## [2,]    7    2    5
## [3,]    5    3    5
```

```r
z <- matrix(y, nrow=3, byrow=T)
# By changing the "byrow" option, we can fill the matrix by row
z
```

```
##      [,1] [,2] [,3]
## [1,]    3    7    5
## [2,]    1    2    3
## [3,]    2    5    5
```

```r
## Extracting values from matrices:
z[2,] # Row
```

```
## [1] 1 2 3
```

```r
z[,3] # Column
```

```
## [1] 5 3 5
```

```r
z[2,3] # Value
```

```
## [1] 3
```

```r
## Create Dataframes:
dat <- as.data.frame(z)
names(dat) <- c("cat", "giraffe","bowlingball")
dat
```

```
##   cat giraffe bowlingball
## 1   3       7           5
## 2   1       2           3
## 3   2       5           5
```

```r
## R has base functions:
mean(y)
```

```
## [1] 3.666667
```

```r
length(y)
```

```
## [1] 9
```

```r
sd(y)
```

```
## [1] 1.936492
```

```r
var(y)
```

```
## [1] 3.75
```

```r
prod(y) # Takes the product of each element in the vector
```

```
## [1] 31500
```

```r
apply(z, 2, mean) # Very useful in avoiding for loops, also has useful cousins sapply and lapply
```

```
## [1] 2.000000 4.666667 4.333333
```

```r
################################################################################
## Brief Introduction to Statistics with R ##
################################################################################
## Getting help
#help.start()   # Opens html help in web browser (if installed)
#help(help) # find help on how to use help
#?help      # Same as above
#help.search("help")    # Find all functions that include the word 'help'

## Reading in your data
getwd()     # What directory are we in?
```

```
## [1] "/Users/breannechryst/Desktop/snar"
```

```r
#setwd("~/Desktop")
## Set working directory to the directory where we put the data
dat <- read.table("http://www.stat.yale.edu/~blc3/IntroR2015/remote_weight.txt", header=T, sep="", row.n
# Read data including headers, data separated by spaces, no row names
ls()            # List all variables stored in memory
```

```
## [1] "dat" "x"   "y"   "z"
```

```r
head(dat) # Shows the first 6 rows of the data
```

```
##   id remote weight gender
## 1  1      5    151      0
## 2  2      7    152      0
## 3  3      3    153      0
## 4  4      2    165      0
## 5  5      5    138      0
## 6  6      0    149      0
```

```r
head(dat, 10) # the first 10 rows of the data
```

```
##    id remote weight gender
## 1   1      5    151      0
## 2   2      7    152      0
## 3   3      3    153      0
## 4   4      2    165      0
## 5   5      5    138      0
## 6   6      0    149      0
## 7   7      5    142      0
## 8   8      9    139      0
## 9   9      1    140      0
## 10 10      8    138      0
```

```r
tail(dat) # last 6 rows of the data
```

```
##      id remote weight gender
## 95   95     30    167      1
## 96   96     28    154      1
## 97   97     29    181      1
## 98   98     34    172      1
## 99   99     23    159      1
## 100 100     25    177      1
```

```r
## Extracting data from the data frame
dim(dat)            # Find out how many rows and columns in the data set
```

```
## [1] 100   4
```

```r
names(dat)          # List all variable names in the dataset
```

```
## [1] "id"     "remote" "weight" "gender"
```

```r
str(dat)           # Look at the structure of your data
```

```
## 'data.frame':    100 obs. of  4 variables:
##  $ id    : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ remote: int  5 7 3 2 5 0 5 9 1 8 ...
##  $ weight: int  151 152 153 165 138 149 142 139 140 138 ...
##  $ gender: int  0 0 0 0 0 0 0 0 0 0 ...
```

```r
# dat               # See the data frame on the screen
dat[1:5,]           # See the first 5 rows
```

```
##   id remote weight gender
## 1  1      5    151      0
## 2  2      7    152      0
## 3  3      3    153      0
## 4  4      2    165      0
## 5  5      5    138      0
```

```r
dat[,"weight"]      # See only the weight column
```

```
##   [1] 151 152 153 165 138 149 142 139 140 138 137 119 140 145 126 142 127
##  [18] 135 149 134 157 141 146 127 143 151 132 149 144 148 119 129 138 165
##  [35] 142 138 137 154 133 139 146 152 149 128 131 136 148 134 137 137 159
##  [52] 167 175 189 179 145 161 181 177 178 170 158 179 192 183 184 169 182
##  [69] 190 195 182 174 184 188 150 178 186 170 149 203 172 160 179 195 160
##  [86] 147 144 178 168 157 184 133 160 170 167 154 181 172 159 177
```

```r
dat[,3]         # Same as above
```

```
##   [1] 151 152 153 165 138 149 142 139 140 138 137 119 140 145 126 142 127
##  [18] 135 149 134 157 141 146 127 143 151 132 149 144 148 119 129 138 165
##  [35] 142 138 137 154 133 139 146 152 149 128 131 136 148 134 137 137 159
##  [52] 167 175 189 179 145 161 181 177 178 170 158 179 192 183 184 169 182
##  [69] 190 195 182 174 184 188 150 178 186 170 149 203 172 160 179 195 160
##  [86] 147 144 178 168 157 184 133 160 170 167 154 181 172 159 177
```

```r
dat$weight          # Yet another way
```

```
##   [1] 151 152 153 165 138 149 142 139 140 138 137 119 140 145 126 142 127
##  [18] 135 149 134 157 141 146 127 143 151 132 149 144 148 119 129 138 165
##  [35] 142 138 137 154 133 139 146 152 149 128 131 136 148 134 137 137 159
##  [52] 167 175 189 179 145 161 181 177 178 170 158 179 192 183 184 169 182
##  [69] 190 195 182 174 184 188 150 178 186 170 149 203 172 160 179 195 160
##  [86] 147 144 178 168 157 184 133 160 170 167 154 181 172 159 177
```

```r
dat[1:5,"weight"]   # See only the first 10 values of the weight col.
```

```
## [1] 151 152 153 165 138
```

```r
#dat[,-1]           # See all but the first column of data
dat.o <- dat        # Copy the data frame to a data.frame named data.O
ls()            # Now we have 5 variables: 'x', 'y', 'z', 'data' and 'data.o'
```

```
## [1] "dat"   "dat.o" "x"     "y"     "z"
```

```r
## Getting familiar with the data
summary(dat)        # Generate summary statistics of data
```

```
##        id             remote          weight          gender
##  Min.   :  1.00   Min.   : 0.00   Min.   :119.0   Min.   :0.0
##  1st Qu.: 25.75   1st Qu.: 5.00   1st Qu.:139.8   1st Qu.:0.0
##  Median : 50.50   Median :14.00   Median :152.0   Median :0.5
##  Mean   : 50.50   Mean   :17.59   Mean   :156.4   Mean   :0.5
##  3rd Qu.: 75.25   3rd Qu.:30.00   3rd Qu.:174.2   3rd Qu.:1.0
##  Max.   :100.00   Max.   :37.00   Max.   :203.0   Max.   :1.0
```

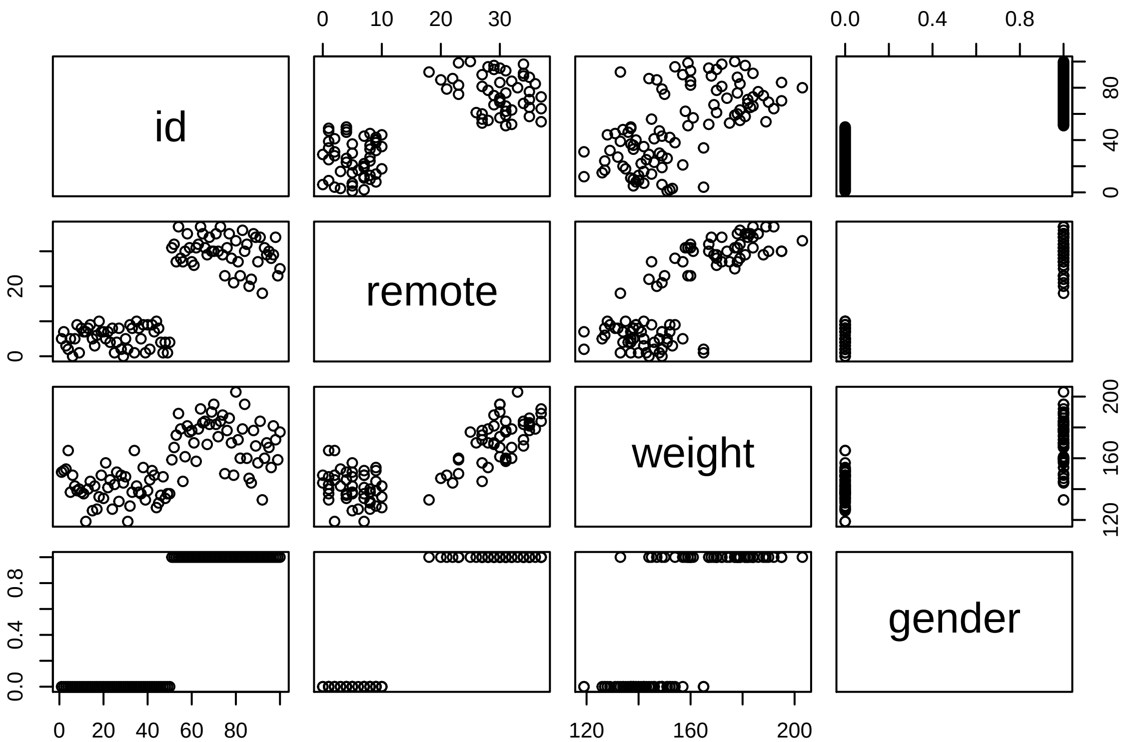
```r
apply(dat, 2, sd)          # Calculate standard deviations of all variables
```

```
##         id     remote     weight     gender
## 29.0114920 12.8488454 20.0833491  0.5025189
```

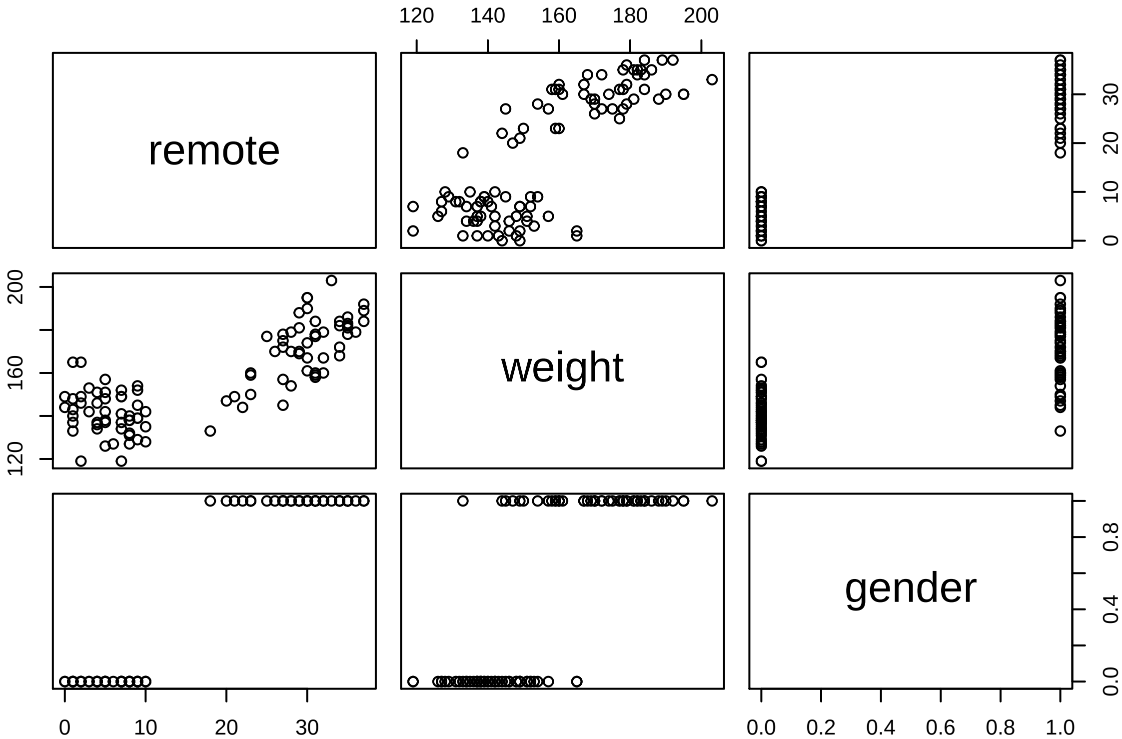
```r
var(dat)         # Variance on diagonal, covariance off diagonal
```

```
##                id     remote     weight     gender
## id      841.66667 299.287879 355.691919 12.6262626
## remote  299.28788 165.092828 209.448990  6.1565657
## weight  355.69192 209.448990 403.340909  7.7929293
## gender   12.62626   6.156566   7.792929  0.2525253
```

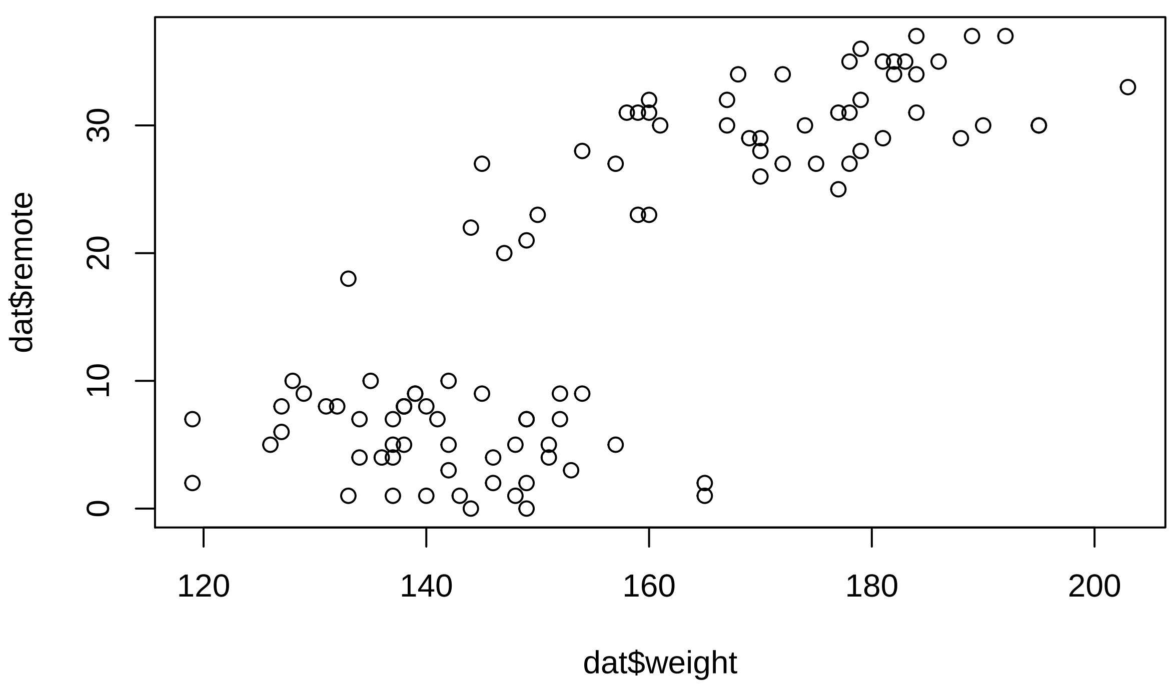
```r
pairs(dat)       # A general view of data through scatter plots
```




```r
pairs(dat[,-1])      # See scatterplots for all pairs of variables except the first ('id') in the data f
```




```r
plot(dat$weight, dat$remote)     # Scatterplot of 'weight' vs. 'remote'
```




```r
# Changing data type
class(dat$gender)        # What kind of variable is 'gender'?
```

```
## [1] "integer"
```

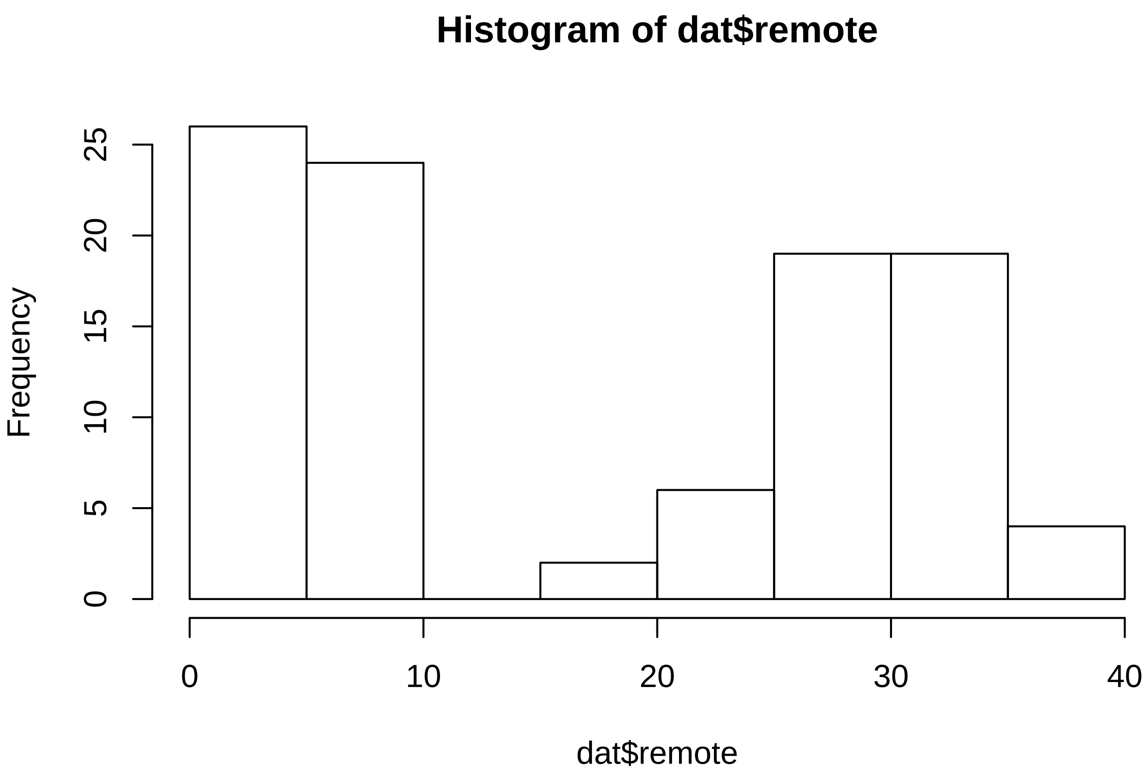
```r
dat$gender <- factor(dat$gender)    # Converts 'gender' from type integer to factor
class(dat$gender)        # Verify that gender is now indeed of type factor
```

```
## [1] "factor"
```

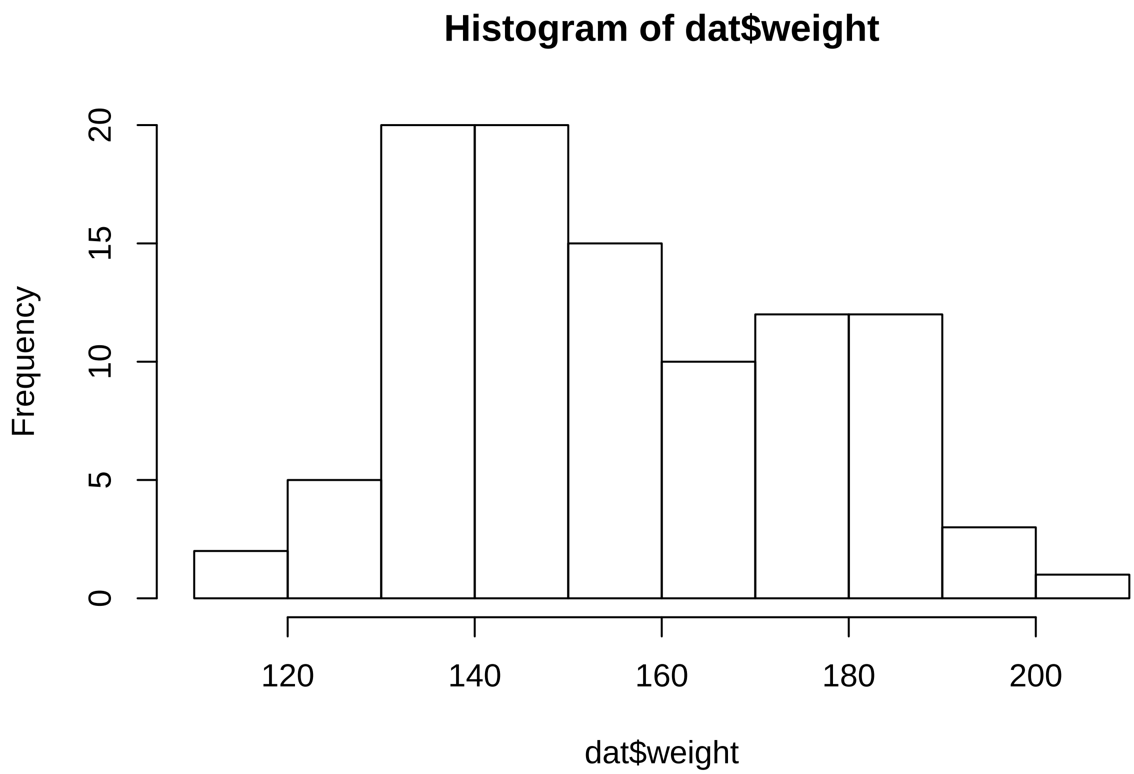
```r
dat$gender          # See all data in column 'gender'; note "Levels: 0 1" at the bottom
```

```
##   [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [36] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [71] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## Levels: 0 1
```

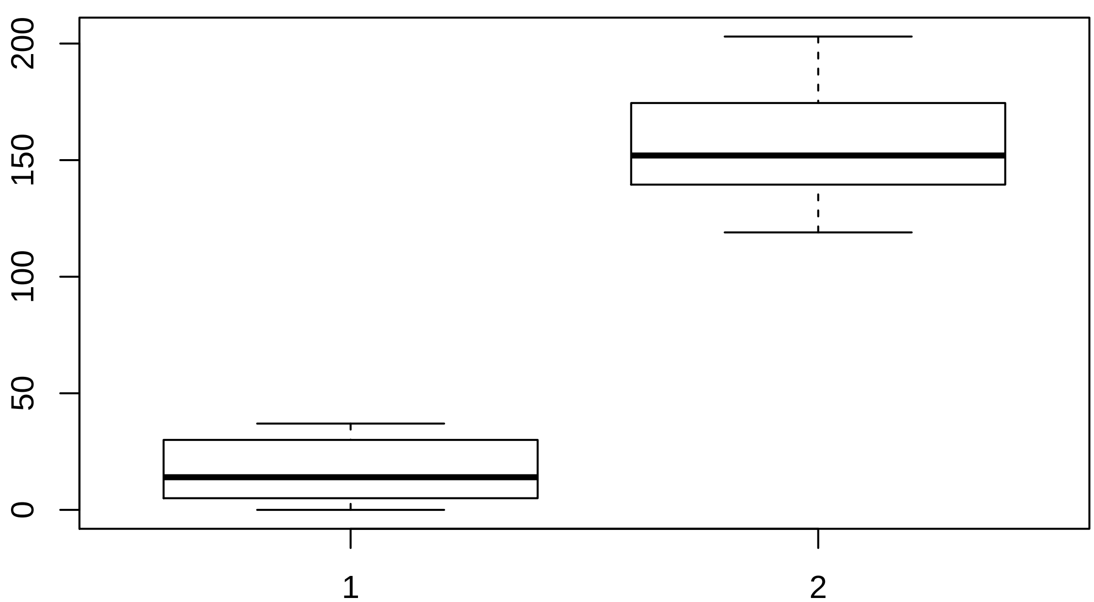
```r
# Basic Graphics
hist(dat$remote)            # Histogram of 'remote'
```



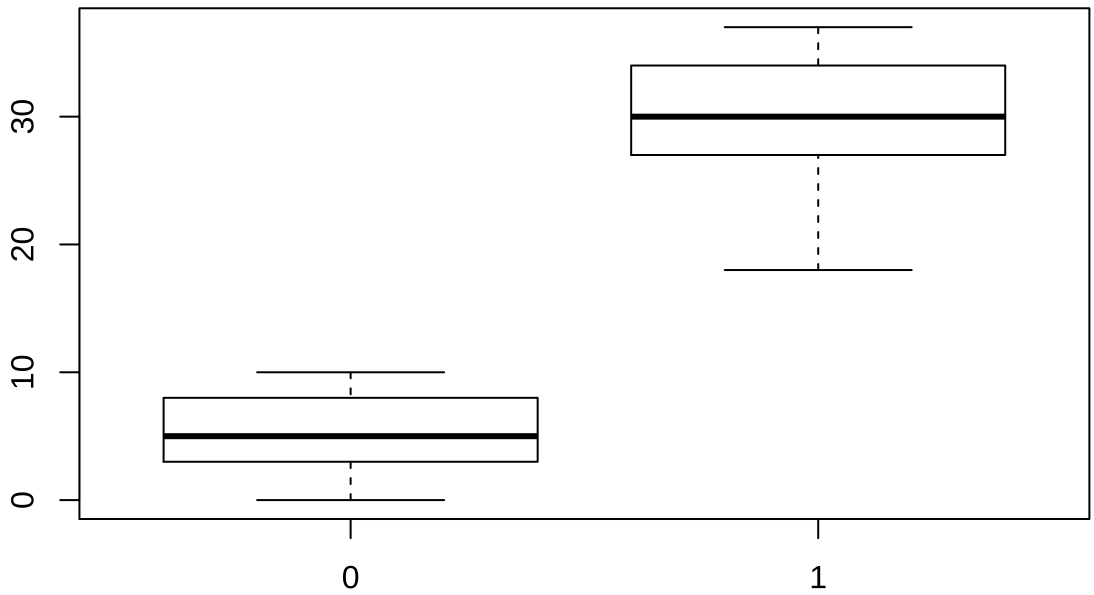

```r
hist(dat$weight)            # Histogram of 'weight'
```



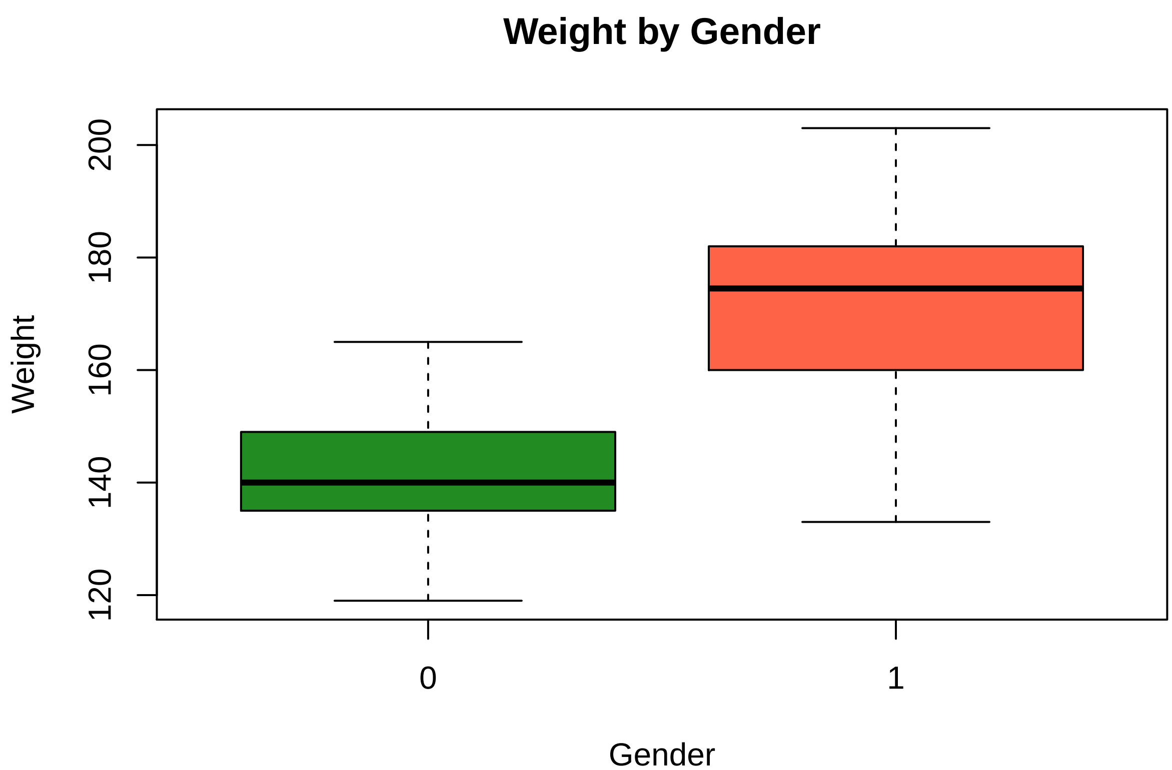

```
boxplot(dat$remote, dat$weight)      # Boxplot of 'remote' and 'weight'
```




```
boxplot(remote ~ gender, data = dat)     # Boxplot of 'remote' conditioned on 'gender'
```




```
boxplot(weight ~ gender, data = dat, main = "Weight by Gender",
        xlab = "Gender", ylab = "Weight",
        col = c("forestgreen", "tomato"))    # Boxplot of 'weight' conditioned on 'gender'
```




```
################################################################################
## Introduction to SNA in R ##
################################################################################
# Installing the packages to be used in the analysis
#install.packages("igraph")
#install.packages("igraphdata")
```

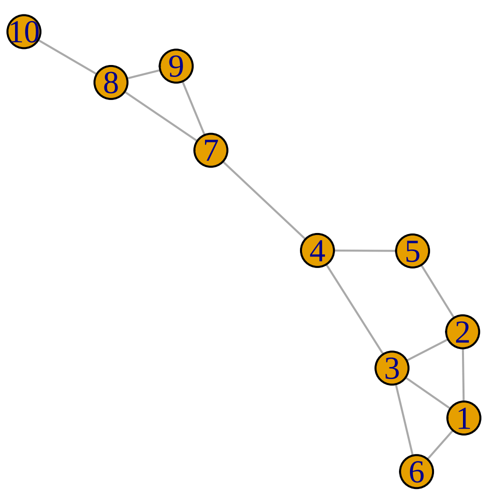
```
# loading package igraph
library(igraph) # functions for igraph: http://igraph.org/r/doc/
```

```
##
## Attaching package: 'igraph'

## The following objects are masked from 'package:stats':
##
##     decompose, spectrum

## The following object is masked from 'package:base':
##
##     union
```

```
# loading package igraphdata
library(igraphdata)

#A Simple Example:
g <- graph(c(1,2, 2,3, 3,4, 4,5, 3,1, 4,7, 2,5,
              6,1, 3,6, 7,8, 7,9,
              8,9, 8,10), directed=F)
g           # summary information
```

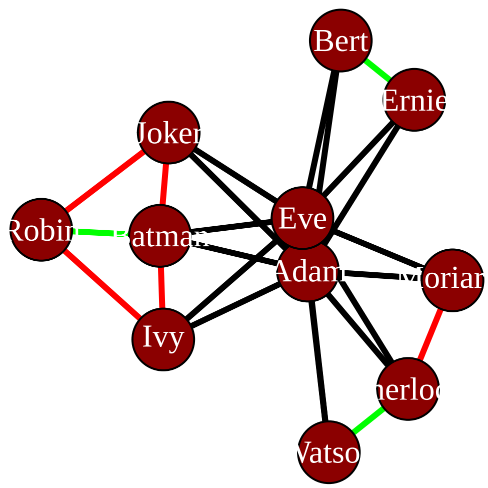
```
## IGRAPH U--- 10 13 --
## + edges:
##  [1] 1-- 2 2-- 3 3-- 4 4-- 5 1-- 3 4-- 7 2-- 5 1-- 6 3-- 6 7-- 8 7-- 9
## [12] 8-- 9 8--10
```

```
plot(g) # network picture
```

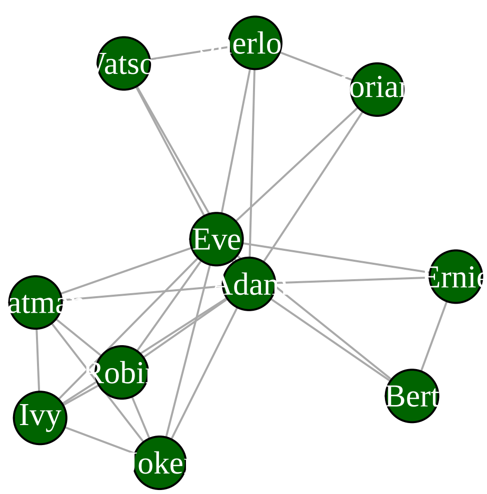



```
################################################################################
## Reading in data and creating plots ##
################################################################################
# Read in the edgelist I made up:
dat <- read.csv("http://www.stat.yale.edu/~blc3/SNA2016/Partners.csv")
```

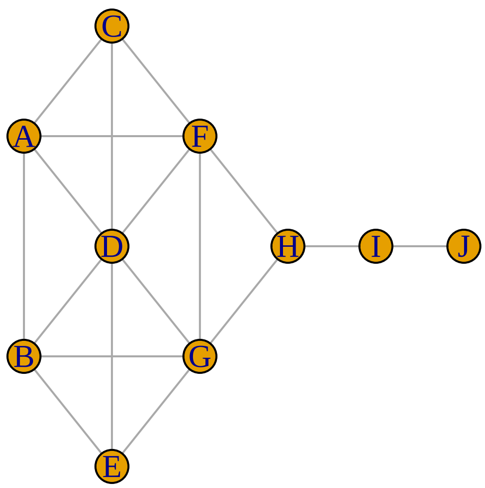
```
# Make a igraph object from the edgelist
partners <-  graph.data.frame(dat, directed=F)

# Add edge colors determined by the variable "Type"
E(partners)$color <- c("black", "red", "green")[as.numeric(dat$Type)]

# Plotting my social network
plot(partners, vertex.size=30, edge.color = E(partners)$color,
     vertex.label.color = "white",
     vertex.color="darkred", edge.width=3 )
```




```
# Read in the adjacency version of the data
dat1 <- read.csv("http://www.stat.yale.edu/~blc3/SNA2016/AdjMat.csv", header = T)
# Create an igraph object from the matrix
g1 <- graph_from_adjacency_matrix(as.matrix(dat1[,-1]), mode = "undirected")
# Plot the network
plot(g1, vertex.size=25,vertex.label.color = "white",
     vertex.color="darkgreen")
```



```r
# Read in the Enron email data and plot
data(enron)
#enron <- delete_vertices(enron, c(72, 118))

#plot(enron, vertex.size=5, vertex.label=NA,
#     edge.arrow.size=.5)

# Read in the kite data and plot
data(kite)
plot(kite)
```




```r
# Read in an existing Media data, first the nodes, then the edges
media.node <- read.csv("http://www.stat.yale.edu/~blc3/SNA2016/Media-NODES.csv")
head(media.node)
```

```
##    id            media media.type type.label audience.size
## 1 s01         NY Times          1  Newspaper            20
## 2 s02            Wa Po          1  Newspaper            25
## 3 s03 Wall St. Journal          1  Newspaper            30
## 4 s04        USA Today          1  Newspaper            32
## 5 s05         LA Times          1  Newspaper            20
## 6 s06          NY Post          1  Newspaper            50
```

```r
media.edge <- read.csv("http://www.stat.yale.edu/~blc3/SNA2016/Media-EDGES.csv")
head(media.edge)
```

```
##   from  to   weight      type
## 1  s01 s02 1.833333 hyperlink
## 2  s01 s02 2.000000 hyperlink
## 3  s01 s03 2.833333 hyperlink
## 4  s01 s04 2.750000 hyperlink
## 5  s04 s11 2.833333   mention
## 6  s05 s15 2.750000   mention
```

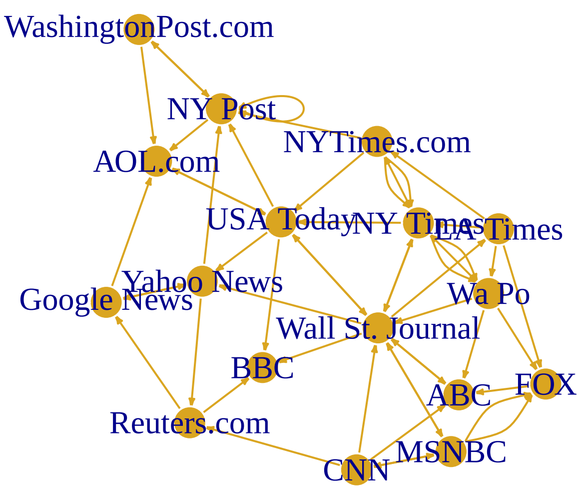
```r
# Make a graph out of the node and edge files
media <- graph.data.frame(media.edge, media.node, directed=T)

# The network object in R
media
```

```
## IGRAPH DNW- 17 52 --
## + attr: name (v/c), media (v/c), media.type (v/n), type.label
## | (v/c), audience.size (v/n), weight (e/n), type (e/c)
## + edges (vertex names):
##  [1] s01->s02 s01->s02 s01->s03 s01->s04 s04->s11 s05->s15 s06->s17
##  [8] s08->s09 s08->s09 s03->s04 s04->s03 s01->s15 s15->s01 s15->s01
## [15] s16->s17 s16->s06 s06->s16 s09->s10 s08->s07 s07->s08 s07->s10
## [22] s05->s02 s02->s03 s02->s01 s03->s01 s12->s13 s12->s14 s14->s13
## [29] s13->s12 s05->s09 s02->s10 s03->s12 s04->s06 s10->s03 s03->s10
## [36] s04->s12 s13->s17 s06->s06 s14->s11 s03->s11 s12->s06 s04->s17
## [43] s17->s04 s08->s03 s03->s08 s07->s14 s15->s06 s15->s04 s05->s01
## + ... omitted several edges
```

```r
# Plot of the network
plot(media, edge.arrow.size=.2, edge.color="goldenrod",
     vertex.color="goldenrod", vertex.frame.color="white",
     vertex.label=V(media)$media)

# extracting adjacency matrix

# full adjacency matrix
get.adjacency(partners, sparse=F)
```

```
##          Bert Adam Batman Sherlock Robin Eve Ernie Joker Watson Ivy
## Bert        0    1      0        0     0   1     1     0      0   0
## Adam        1    0      1        1     0   1     1     1      1   1
## Batman      0    1      0        0     1   1     0     1      0   1
## Sherlock    0    1      0        0     0   1     0     0      1   0
## Robin       0    0      1        0     0   0     0     1      0   1
## Eve         1    1      1        1     0   0     1     1      1   1
## Ernie       1    1      0        0     0   1     0     0      0   0
## Joker       0    1      1        0     1   1     0     0      0   0
## Watson      0    1      0        1     0   1     0     0      0   0
## Ivy         0    1      1        0     1   1     0     0      0   0
## Moriarty    0    1      0        1     0   1     0     0      0   0
##          Moriarty
## Bert            0
## Adam            1
## Batman          0
## Sherlock        1
## Robin           0
## Eve             1
## Ernie           0
## Joker           0
## Watson          0
## Ivy             0
## Moriarty        0
```

```r
get.adjacency(partners, sparse=T)
```




```
## 11 x 11 sparse Matrix of class "dgCMatrix"

##    [[ suppressing 11 column names 'Bert', 'Adam', 'Batman' ... ]]

##
## Bert     . 1 . . . 1 1 . . . .
## Adam     1 . 1 1 . 1 1 1 1 1 1
## Batman   . 1 . . 1 1 . 1 . 1 .
## Sherlock . 1 . . . 1 . . 1 . 1
## Robin    . . 1 . . . . 1 . 1 .
## Eve      1 1 1 1 . . 1 1 1 1 1
## Ernie    1 1 . . . 1 . . . . .
## Joker    . 1 1 . 1 1 . . . . .
## Watson   . 1 . 1 . 1 . . . . .
## Ivy      . 1 1 . 1 1 . . . . .
## Moriarty . 1 . 1 . 1 . . . . .
```

```r
# only upper triangle ('g' is undirected)
get.adjacency(partners, type="upper", sparse=FALSE)
```

```
##          Bert Adam Batman Sherlock Robin Eve Ernie Joker Watson Ivy
## Bert        0    1      0        0     0   1     1     0      0   0
## Adam        0    0      1        1     0   1     1     1      1   1
## Batman      0    0      0        0     1   1     0     1      0   1
## Sherlock    0    0      0        0     0   1     0     0      1   0
## Robin       0    0      0        0     0   0     0     1      0   1
## Eve         0    0      0        0     0   0     1     1      1   1
## Ernie       0    0      0        0     0   0     0     0      0   0
## Joker       0    0      0        0     0   0     0     0      0   0
## Watson      0    0      0        0     0   0     0     0      0   0
## Ivy         0    0      0        0     0   0     0     0      0   0
## Moriarty    0    0      0        0     0   0     0     0      0   0
##          Moriarty
## Bert            0
```

```
## Adam            1
## Batman          0
## Sherlock        1
## Robin           0
## Eve             1
## Ernie           0
## Joker           0
## Watson          0
## Ivy             0
## Moriarty        0
```

```r
# extracting edgelist
get.edgelist(partners)
```

```
##       [,1]       [,2]
##  [1,] "Bert"     "Ernie"
##  [2,] "Adam"     "Eve"
##  [3,] "Batman"   "Joker"
##  [4,] "Sherlock" "Watson"
##  [5,] "Batman"   "Robin"
##  [6,] "Batman"   "Ivy"
##  [7,] "Robin"    "Ivy"
##  [8,] "Robin"    "Joker"
##  [9,] "Sherlock" "Moriarty"
## [10,] "Adam"     "Batman"
## [11,] "Adam"     "Joker"
## [12,] "Adam"     "Sherlock"
## [13,] "Adam"     "Watson"
## [14,] "Adam"     "Ivy"
## [15,] "Adam"     "Moriarty"
## [16,] "Batman"   "Eve"
## [17,] "Eve"      "Joker"
## [18,] "Sherlock" "Eve"
## [19,] "Eve"      "Watson"
## [20,] "Eve"      "Ivy"
## [21,] "Eve"      "Moriarty"
## [22,] "Bert"     "Adam"
## [23,] "Adam"     "Ernie"
## [24,] "Bert"     "Eve"
## [25,] "Eve"      "Ernie"
```

```r
get.edgelist(kite)
```

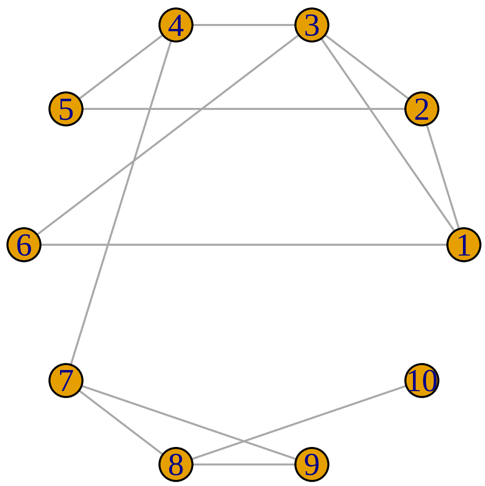
```
##      [,1] [,2]
## [1,] "A"  "B"
## [2,] "A"  "C"
## [3,] "A"  "D"
## [4,] "A"  "F"
## [5,] "B"  "D"
## [6,] "B"  "E"
## [7,] "B"  "G"
## [8,] "C"  "D"
## [9,] "C"  "F"
```

```
## [10,] "D"  "E"
## [11,] "D"  "F"
## [12,] "D"  "G"
## [13,] "E"  "G"
## [14,] "F"  "G"
## [15,] "F"  "H"
## [16,] "G"  "H"
## [17,] "H"  "I"
## [18,] "I"  "J"
```

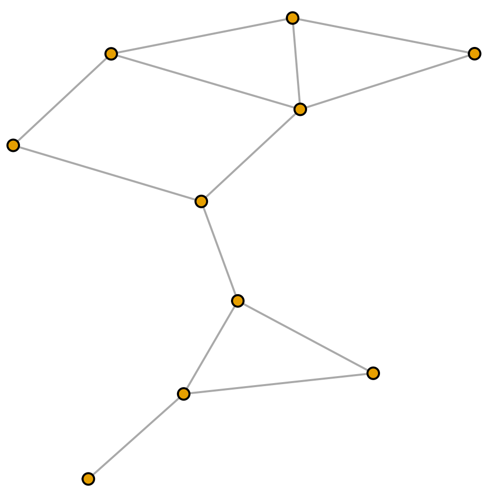
```
# Setting edge attributes:
E(partners)$weight <- runif(25,1,5)

# Setting vertex attributes:
V(partners)$gender <- c("M", "M", "M", "F", "M", "M", "M", "M", "M", "F", "M")
V(partners)$color= ifelse(V(partners)$gender == "M", "tomato", "gold")

# Network object with new attributes
partners
```

```
## IGRAPH UNW- 11 25 --
## + attr: name (v/c), gender (v/c), color (v/c), Weight (e/n), Type
## | (e/c), color (e/c), weight (e/n)
## + edges (vertex names):
##  [1] Bert    --Ernie    Adam    --Eve      Batman  --Joker
##  [4] Sherlock--Watson   Batman  --Robin    Batman  --Ivy
##  [7] Robin   --Ivy      Robin   --Joker    Sherlock--Moriarty
## [10] Adam    --Batman   Adam    --Joker    Adam    --Sherlock
## [13] Adam    --Watson   Adam    --Ivy      Adam    --Moriarty
## [16] Batman  --Eve      Eve     --Joker    Sherlock--Eve
## [19] Eve     --Watson   Eve     --Ivy      Eve     --Moriarty
## + ... omitted several edges
```

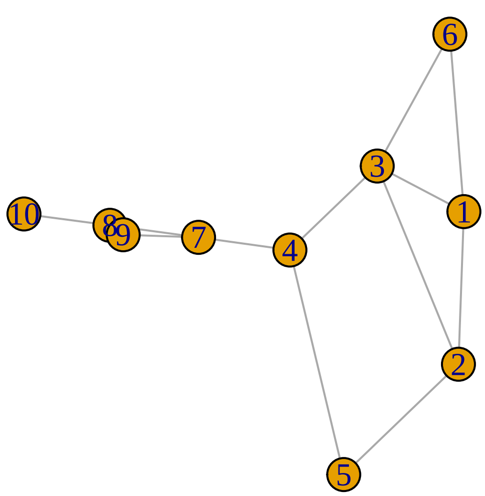
```
################################################################################
## Introduction to Social Network Visualization in R ##
################################################################################
# see ?igraph.plotting for detailed explanation of all
# the options

## Layouts
# some available layouts
# Default is Fruchterman-Reingold
# circle layout
plot(g, layout=layout.circle)
```



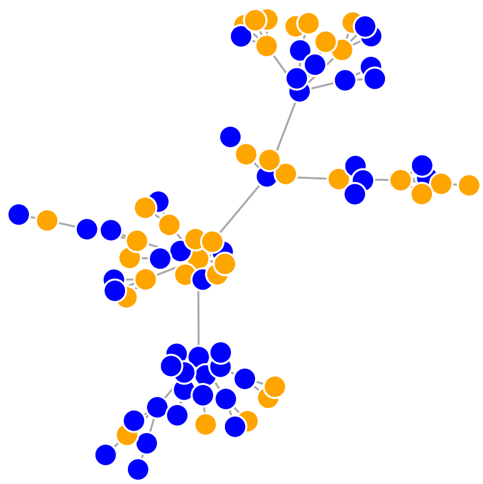

```
# Kamada-Kawai
plot(g, layout=layout.kamada.kawai,
     vertex.label=NA, vertex.size=5, edge.arrow.size=0.5)
```



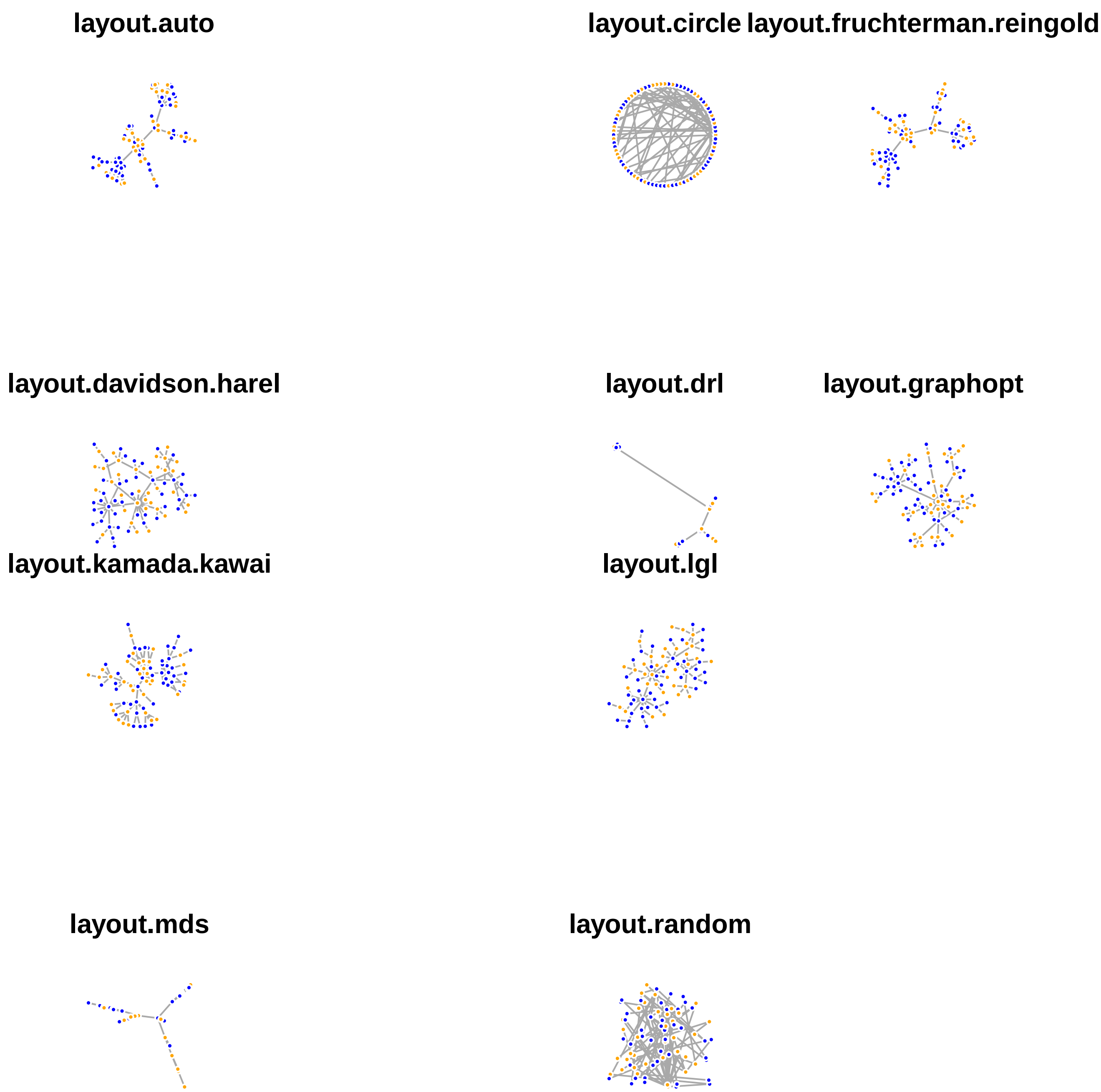
```
# Multidimensional Scaling
plot(g, layout=layout.mds)
```



```r
# Simulate a random network for visualiation:
net.bg <- barabasi.game(80)

# Setting node and edge attributes for nice graphs
V(net.bg)$frame.color <- "white"
V(net.bg)$color <- sample(c("orange", "blue"),80, replace = T)
V(net.bg)$label <- ""
V(net.bg)$size <- 10
E(net.bg)$arrow.mode <- 0

# Base plot
plot(net.bg)
```

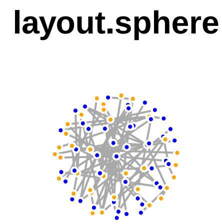


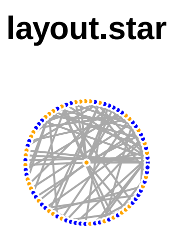
```r
# All the possible layout options in R
layouts <- grep("^layout\\.", ls("package:igraph"), value=TRUE)

# Remove layouts that do not apply to our graph.
layouts <- layouts[!grepl("bipartite|merge|norm|sugiyama|spring|grid.3d|svd|fruchterman.reingold.grid",

# Setting the graph window to a 2 by 2
par(mfrow=c(2,2))

# For loop to plot our simulated network under several layouts
for (layout in layouts) {
  l <- do.call(layout, list(net.bg))
  plot(net.bg, edge.arrow.mode=0, layout=l, main=layout) }
```



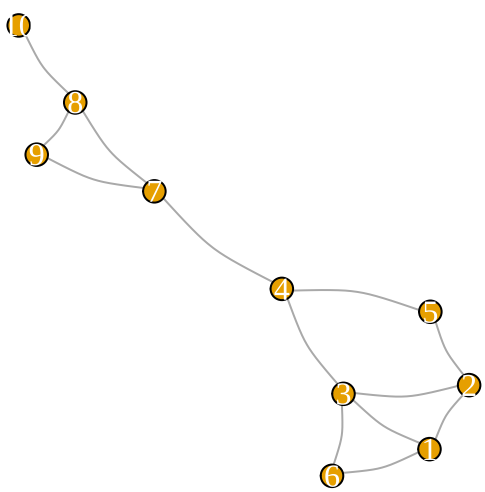

```r
# Resetting graph window to contain just one plot
par(mfrow=c(1,1))
```

layout.reingold.tilford

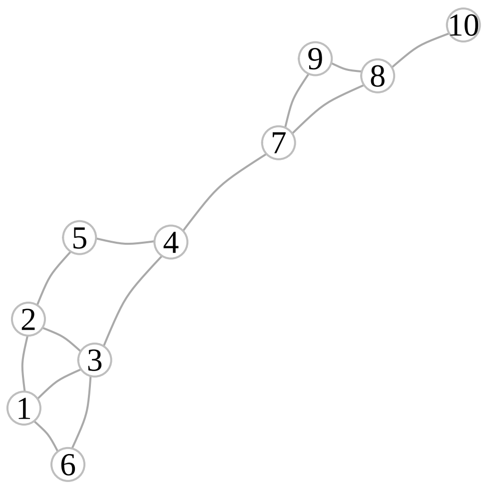




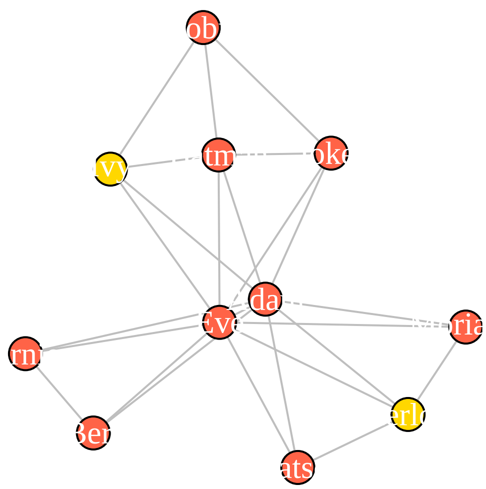

```r
### Network plotting examples
# Curved edges
# 'edge.curved' can be between 0 and 1
plot(g, vertex.label.color="white", edge.curved=0.2,
     edge.arrow.size=0.5, vertex.size=10)
```




```r
# vertex frames color
plot(g, vertex.label.color="black",
     vertex.frame.color="gray", vertex.color="white",
     edge.curved=0.2, edge.arrow.size=.5)
```



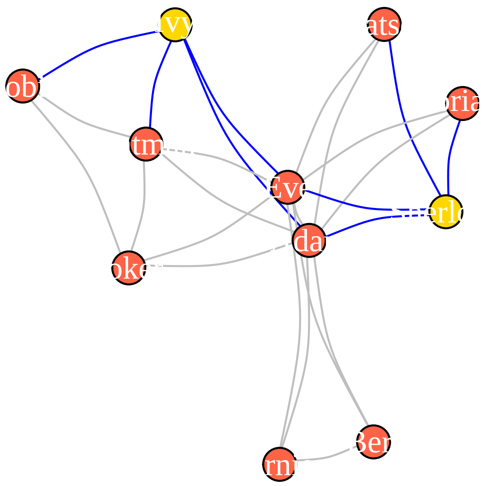

```r
# Vertex and edge attributes with names including
# "color", "label", "size", etc. are used by the plotting
# function. Instead of specifying an argument to 'plot'
# we can set an appropriate attribute.
#
# Example: Highlight ties involving at least one Female

# default edge color
E(partners)$color <- "gray"

# verify
plot(partners, edge.arrow.size=.5,
     vertex.label.color="white")
```



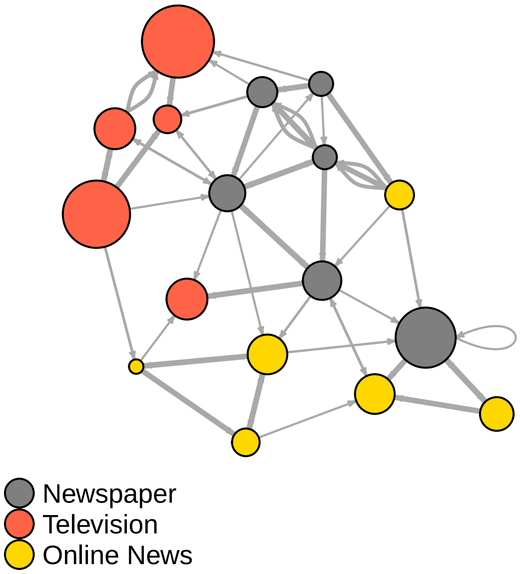
```r
# set attribute 'color' for edges sent by females
female.ids <- V(partners)[gender=="F"]
female.ids
```

```
## + 2/11 vertices, named:
## [1] Sherlock Ivy
```

```r
E(partners)[from(female.ids)]$color <- "blue"

plot(partners, edge.arrow.size=.5,
     vertex.label.color="white", edge.curved=0.2)
```



```r
## Media graph
# Generate colors base on media type:
colors <- c("gray50", "tomato", "gold")
V(media)$color <- colors[V(media)$media.type]

# Use the audience size value for the node size:
V(media)$size <- V(media)$audience.size*0.6

# The labels are currently node IDs.
# Setting them to NA will render no labels:
V(media)$label <- NA

#change arrow size and edge color:
E(media)$arrow.size <- .2
E(media)$edge.color <- "gray80"
E(media)$width <- E(media)$weight
plot(media)
legend(x=-1.5, y=-1.1, c("Newspaper","Television", "Online News"), pch=21,
       pt.bg=colors, pt.cex=2, cex=.8, bty="n", ncol=1)
```




```r
#####################################################################################
## Introduction to Social Network Statistics in R ##
#####################################################################################
# Media network
vcount(media) # number of nodes
```

```
## [1] 17
```

```r
ecount(media) # number of edges
```

```
## [1] 52
```

```r
graph.density(media)  # density (very sparse)
```

```
## [1] 0.1911765
```

```r
# Enron network
vcount(enron)
```

```
## [1] 184
```

```r
ecount(enron)
```

```
## [1] 125409
```

```r
graph.density(enron, loops=TRUE)
```

```
## [1] 3.704188
```

```r
# Kite network
vcount(kite)
```

```
## [1] 10
```

```r
ecount(kite)
```

```
## [1] 18
```

```r
graph.density(kite, loops=TRUE)
```

```
## [1] 0.3272727
```

```r
### Degrees and their distribution ###
# vector of degrees (directed graph)
head(degree(enron))  # total degree
```

```
## [1] 114 428 391 104 957 381
```

```r
head(degree(enron, mode="in"))   # in-degree
```

```
## [1]  78 335 224  88 623 211
```

```r
head(degree(enron, mode="out"))  # out-degree
```

```
## [1]  36  93 167  16 334 170
```

```r
summary(degree(enron))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     2.0   220.5   538.0  1363.0  1556.0 21560.0
```

```r
# vector of degrees (undirected graph)
degree(partners)  # total degree
```

```
##     Bert     Adam   Batman Sherlock    Robin      Eve    Ernie    Joker
##        3        9        5        4        3        9        3        4
##   Watson      Ivy Moriarty
##        3        4        3
```

```r
degree(partners, mode="in")   # in-degree
```

```
##     Bert     Adam   Batman Sherlock    Robin      Eve    Ernie    Joker
##        3        9        5        4        3        9        3        4
##   Watson      Ivy Moriarty
##        3        4        3
```

```r
degree(partners, mode="out")  # out-degree
```

```
##     Bert     Adam   Batman Sherlock    Robin      Eve    Ernie    Joker
##        3        9        5        4        3        9        3        4
##   Watson      Ivy Moriarty
##        3        4        3
```

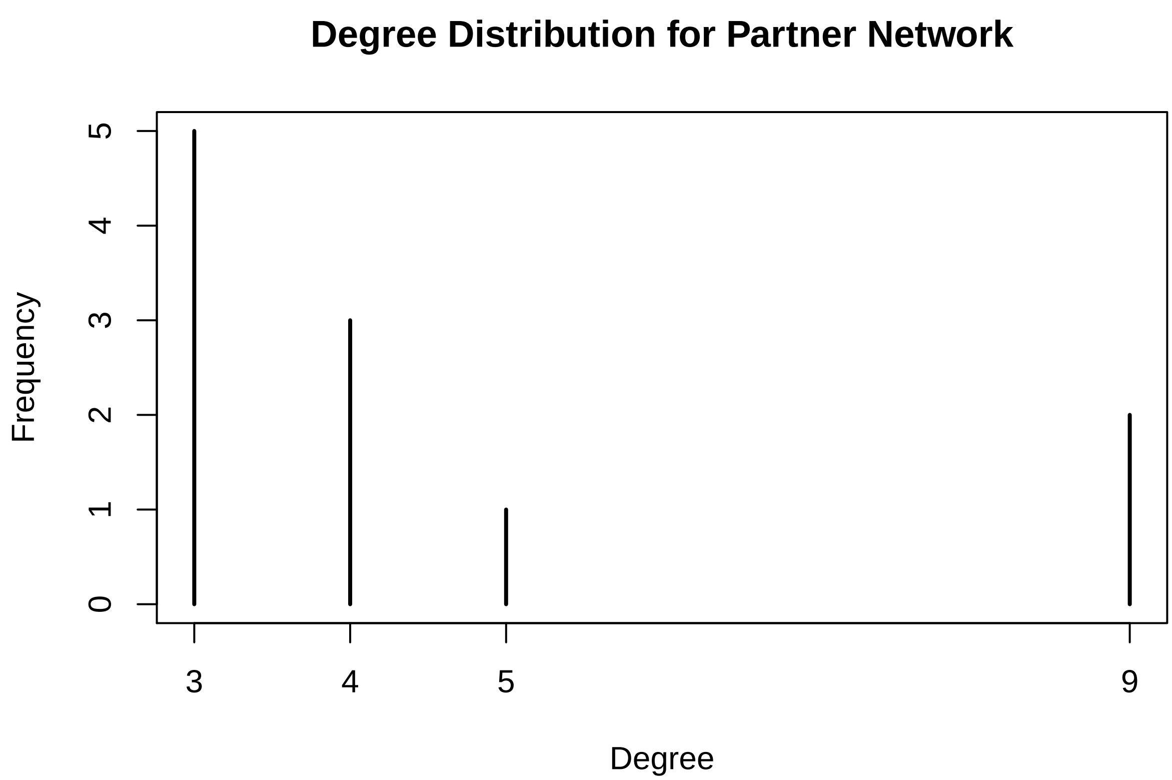
```r
# for undirected graphs the in and out-degrees are the same

summary(degree(partners))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   3.000   3.000   4.000   4.545   4.500   9.000
```

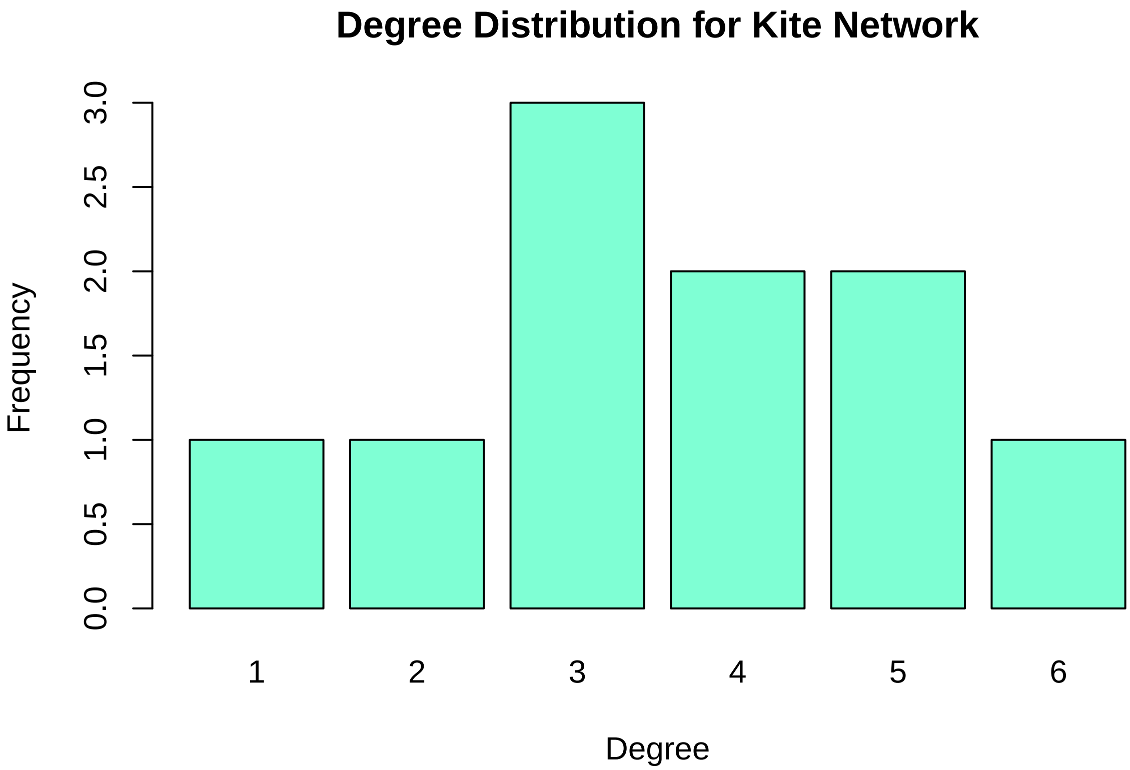
```r
# degree distribution
degtab <- table(degree(partners))
degtab
```

```
##
## 3 4 5 9
## 5 3 1 2
```

```r
plot(degtab, main = "Degree Distribution for Partner Network", xlab = "Degree", ylab = "Frequency") # p
```



```r
barplot(table(degree(kite)), main = "Degree Distribution for Kite Network", xlab = "Degree", ylab = "Fre
```



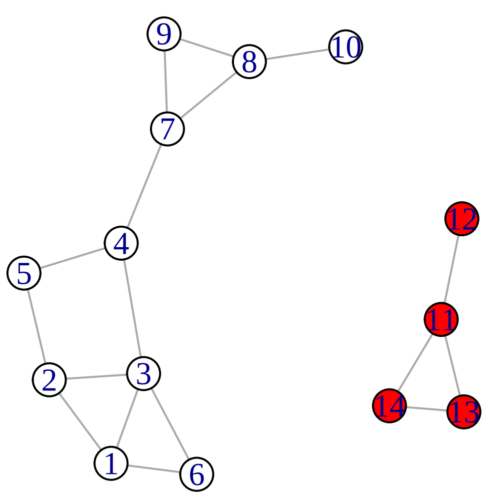

```r
## Subgraphs and components ###
## Components (clusters)
k <- clusters(enron)
k
```

```
## $membership
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [36] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [71] 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [106] 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [141] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [176] 1 1 1 1 1 1 1 1 1
##
## $csize
## [1] 182   1   1
##
## $no
## [1] 3
```

```r
# 'k' is a list with:
# membership  =  vector assigning nodes to components
# component size = number of nodes in each component
# number of components

# "strong" = relationships have to go both ways
k2 <- clusters(enron, "strong")
k2
```

```
## $membership
##   [1]  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4
##  [24]  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4 11  4  4  4
##  [47]  4  4  4  4  4  4  7  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4
##  [70]  4  4  3  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  6  4  4  4  4
##  [93]  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4 10  4  4  4
## [116]  4  4  2  4  4  4  4  8  4  4  4  4  4  4  4  4  4  4  4  4  1  4  4
## [139]  4  4  4  4  4  4  4  4  4  4  4  4  5  4  4  4  4  4  4  4  4  4  4
## [162]  4  4  4  9  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4
##
## $csize
##  [1]   1   1   1 174   1   1   1   1   1   1   1
##
## $no
## [1] 11
```

```r
# Create a new graph for illuatration
k <- g %>%
  add_vertices(4, color = "red") %>%
  add_edges(c(13,14, 11,12, 11,13, 11,14))
plot(k)
```




```r
k3 <- clusters(k, "strong")

### Paths ###

# Matrix of shortest paths between nodes
shortest.paths(kite)
```

```
##   A B C D E F G H I J
## A 0 1 1 1 2 1 2 2 3 4
## B 1 0 2 1 1 2 1 2 3 4
## C 1 2 0 1 2 1 2 2 3 4
## D 1 1 1 0 1 1 1 2 3 4
## E 2 1 2 1 0 2 1 2 3 4
## F 1 2 1 1 2 0 1 1 2 3
## G 2 1 2 1 1 1 0 1 2 3
```

```
## H 2 2 2 2 2 1 1 0 1 2
## I 3 3 3 3 3 2 2 1 0 1
## J 4 4 4 4 4 3 3 2 1 0
```

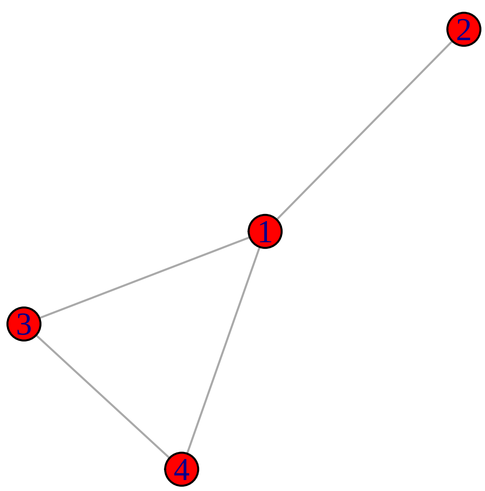
```r
average.path.length(kite)
```

```
## [1] 1.977778
```

```r
### Subgraphs ###

# Create subgraph containing all nodes in the largest
# strongly connected component.
# Using 'induced.subgraph'.

which.max(k3$csize)
```

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

```r
i <- which(k3$membership == which.min(k3$csize))
# largest component of the Enron data
k.lc <- induced.subgraph(k, V(k)[i])

plot(k.lc)
```




```r
### Network diameter ###
# diameter: longest shortest path
# by default directed
diameter(g)
```

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

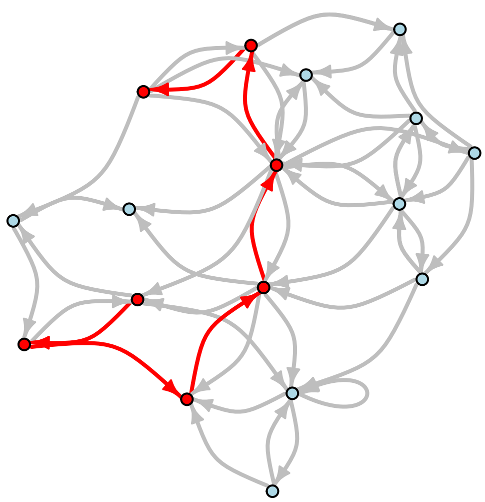
```r
diameter(enron, directed=FALSE)
```

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

```
# get vertex ids of nodes on the longest shortest path
l <- get.diameter(media)
l
```

```
## + 7/17 vertices, named:
## [1] s12 s13 s17 s04 s03 s08 s07
```

```
# color the edges adjacent to these vertices (color the
# shortest path itself)
E(media)$color <- "gray"
E(media, path=l )$color <- "red"
# color the vertices on the path
V(media)$color <- "lightblue"
V(media)[l]$color <- "red"

plot(media, vertex.size=5, vertex.label=NA,
     edge.width=2, edge.arrow.size=0.5,
     edge.curved=0.5)
```



```
### Centrality ###
# vector of betweenness centrality scores
b <- betweenness(g)
b
```

```
##  [1]  0.8333333  2.1666667 13.6666667 20.8333333  2.5000000  0.0000000
##  [7] 18.0000000  8.0000000  0.0000000  0.0000000
```

```
which.max(betweenness(kite))
```

```
## H
## 8
```

```
# eigenvector centralities
ec <- evcent(g)
str(ec)
```

```
## List of 3
##  $ vector : num [1:10] 0.836 0.802 1 0.64 0.495 ...
##  $ value  : num 2.91
##  $ options:List of 20
##   ..$ bmat   : chr "I"
##   ..$ n      : int 10
##   ..$ which  : chr "LA"
##   ..$ nev    : int 1
##   ..$ tol    : num 0
##   ..$ ncv    : int 0
##   ..$ ldv    : int 0
##   ..$ ishift : int 1
##   ..$ maxiter: int 1000
##   ..$ nb     : int 1
##   ..$ mode   : int 1
##   ..$ start  : int 1
##   ..$ sigma  : num 0
##   ..$ sigmai : num 0
##   ..$ info   : int 0
##   ..$ iter   : int 10
##   ..$ nconv  : int 1
##   ..$ numop  : int 32
##   ..$ numopb : int 0
##   ..$ numreo : int 24
```

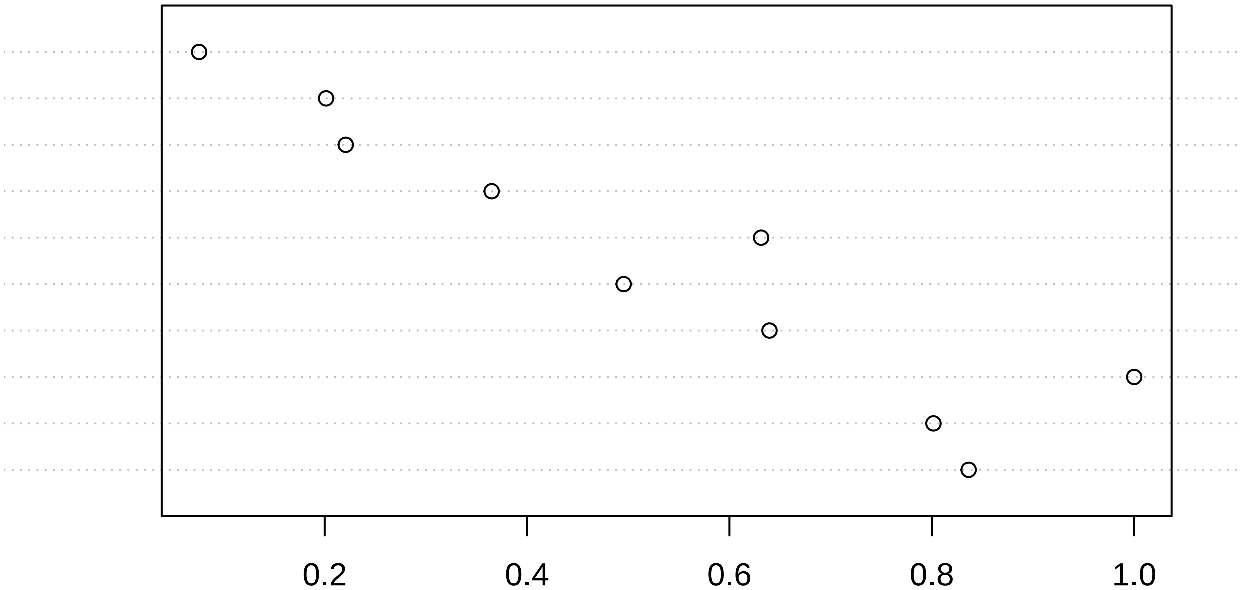
```
ec <- ec$vector
ec
```

```
##  [1] 0.83637756 0.80162320 1.00000000 0.63958076 0.49544949 0.63130019
##  [7] 0.36501528 0.22081267 0.20139284 0.07590981
```

```
which.max(evcent(kite)$vector)
```

```
## D
## 4
```

```
# Dotchart is a barplot alternative
dotchart(ec)
```




```r
# closeness
closeness(g)
```

```
##  [1] 0.04347826 0.04347826 0.05555556 0.06250000 0.04761905 0.04000000
##  [7] 0.05555556 0.04347826 0.04166667 0.03225806
```

```r
head(closeness(enron))
```

```
## [1] 0.0008561644 0.0010526316 0.0009970090 0.0009569378 0.0011013216
## [6] 0.0010626993
```

```r
which.max(closeness(kite))
```

```
## F
## 6
```

```r
### Homophily or assortivity ###
# by degree
assortativity_degree(enron)
```

```
## [1] 0.6762668
```

```r
# by media type
assortativity_nominal(media, V(media)$media.type, directed=F)
```

```
## [1] 0.1990521
```

```r
# by audience size
assortativity(media, V(media)$audience.size, directed=F)
```

```
## [1] -0.01803453
```

```r
### Transitivity ###
transitivity(g)
```

```
## [1] 0.375
```

```r
transitivity(enron)
```

```
## [1] 0.3725138
```

```r
transitivity(kite)
```

```
## [1] 0.5789474
```