Graph Mining

GRAPH ANALYSIS AND VISUALIZATION WITH R

Instructor: Lê Ngọc Thành

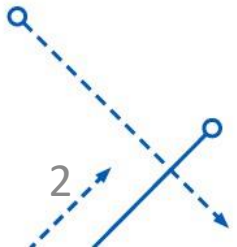
Email: lnthanh@fit.hcmus.edu.vn

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
TP. HỒ CHÍ MINH

fit@hcmus

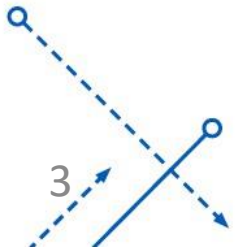
Content

- **Introduction to R language and its syntaxes**
- Vector, Factor, Matrix, List, DataFrame in R
- Plot visualization with R
- Create and generate graph
- Read and save graph data
- Visualization with igraph
- Describe features of graph



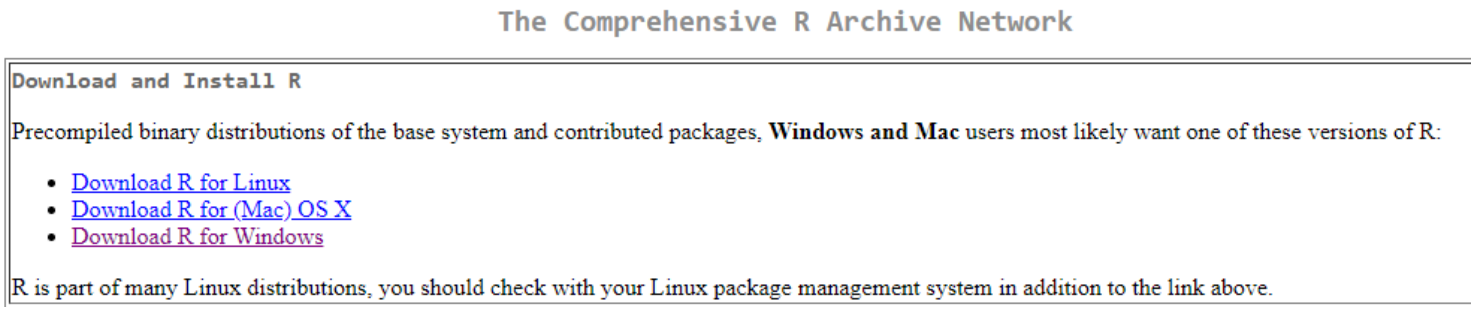
R language

- Created by two statisticians Ross Ihaka and Robert Gentleman (1993)
- Used for various purposes:
  - Computing
  - Dealing with matrix
  - Statistics analysis
- R is a programming language so it can be used to develop software for a specific problem.

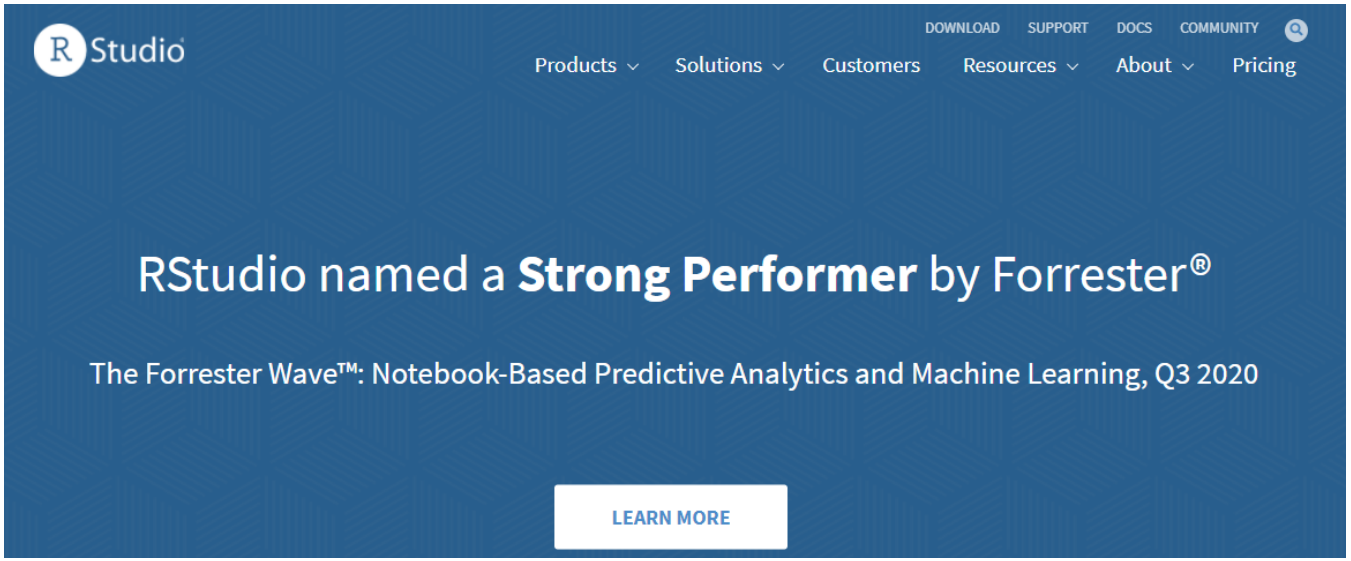


Setting R

- Link for downloading R: https://cran.r-project.org/



- IDE for R: https://rstudio.com/



Simple syntaxes in R

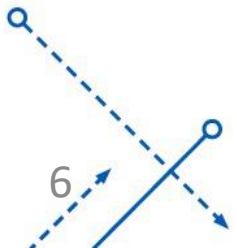
- Assignment statement: $assign()$, <-, $=$

```
x <- 3          # Assignment

x               # Evaluate the expression and print result



y <- 4          # Assignment

y + 5           # Evaluation, y remains 4



z <- x + 17*y   # Assignment

z               # Evaluation
```

Simple syntaxes in R

- Constant:
  - **NA**: undefined data (not available/missing value)
  - **NULL:** null object (null/null list)
  - **Inf** and **-Inf**: positive and negative infinity
  - **NaN**: invalid output (Not a Number)

```
# NA - missing or undefined data

5 + NA      # When used in an expression, the result is generally NA

is.na(5+NA) # Check if missing



# NULL - an empty object, e.g. a null/empty list

10 + NULL     # use returns an empty object (length zero)

is.null(NULL) # check if NULL
```

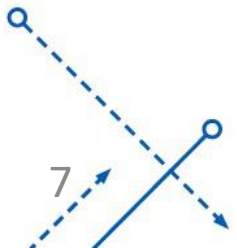


Simple syntaxes in R

- IF-ELSE statement:

```
# if (condition) expr1 else expr2

x <- 5; y <- 10

if (x==0) y <- 0 else y <- y/x #

y
```

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



Simple syntaxes in R

- For loop:

```r
# for (variable in sequence) expr

ASum <- 0; AProd <- 1

for (i in 1:x)

{

  ASum <- ASum + i

  AProd <- AProd * i

}

ASum  # equivalent to sum(1:x)
```

```
## [1] 15
```

```r
AProd # equivalemt to prod(1:x)
```

```
## [1] 120
```

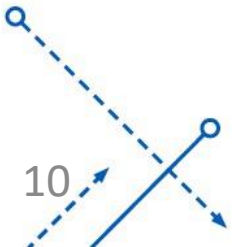
Simple syntaxes in R

- While loop and repeat:

```r
# while (condintion) expr

while (x > 0) {print(x); x <- x-1;}



# repeat expr, use break to exit the loop

repeat { print(x); x <- x+1; if (x>10) break}
```

Nội dung

- **Introduction to R language and its syntaxes**
- **Vector, Factor, Matrix, List, DataFrame in R**
- Plot visualization with R
- Create and generate graph
- Read and save graph data
- Visualization with igraph
- Describe features of graph



Vector in R

- Define vector:

```r
v1 <- c(1, 5, 11, 33)       # Numeric vector, length 4

v2 <- c("hello","world")    # Character vector, length 2 (a vector of string

v3 <- c(TRUE, TRUE, FALSE)  # Logical vector, same as c(T, T, F)

v4 <- c(v1,v2,v3,"boo")     # All elements turn into strings
v <- 1:7          # same as c(1,2,3,4,5,6,7)

v <- rep(0, 77)  # repeat zero 77 times: v is a vector of 77 zeroes

v <- rep(1:3, times=2) # Repeat 1,2,3 twice

v <- rep(1:10, each=2) # Repeat each element twice

v <- seq(10,20,2) # sequence: numbers between 10 and 20, in jumps of 2

v1 <- 1:5          # 1,2,3,4,5

v2 <- rep(1,5)     # 1,1,1,1,1
```

Vector in R

- Access elements in vector:
  - Indexes of vector in R start from 1

```
v1[3]             # third element of v1

v1[2:4]           # elements 2, 3, 4 of v1

v1[c(1,3)]        # elements 1 and 3 - note that your indexes are a vector

v1[c(T,T,F,F,F)]  # elements 1 and 2 - only the ones that are TRUE

v1[v1>3]          # v1>3 is a logical vector TRUE for elements >3
```

Vector in R

- Compare vector:

```
v1 > 2        # Each element is compared to 2, returns logical vector

v1==v2        # Are corresponding elements equivalent, returns logical vector.

v1!=v2        # Are corresponding elements *not* equivalent? Same as !(v1==v2)

(v1>2) | (v2>0)    # | is the boolean OR, returns a vector.

(v1>2) & (v2>0)    # & is the boolean AND, returns a vector.

(v1>2) || (v2>0)   # || is the boolean OR, returns a single value

(v1>2) && (v2>0)   # && is the boolean AND, ditto
```

- Length of vector:

```
length(v1)
```

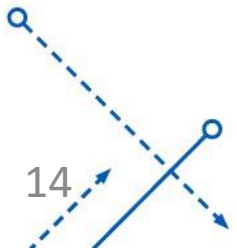
Vector in R

- Manipulating with elements in vector:

```
v1 + v2      # Element-wise addition

v1 + 1       # Add 1 to each element

v1 * 2       # Multiply each element by 2

v1 + c(1,7)  # This doesn't work: (1,7) is a vector of different length
```

- Math operations:

```
sum(v1)      # The sum of all elements

mean(v1)     # The average of all elements

sd(v1)       # The standard deviation

cor(v1,v1*5) # Correlation between v1 and v1*5
```



Factor in R

- Factor is a type of vector which is used to contain categorized data (string).
  - Apart from vector of string, it just save levels for differentiating and store data as integers.

```
eye.col.v <- c("brown", "green", "brown", "blue", "blue", "blue")         #vector

eye.col.f <- factor(c("brown", "green", "brown", "blue", "blue", "blue")) #factor

eye.col.v
```

```
## [1] "brown" "green" "brown" "blue"  "blue"  "blue"
```

```
eye.col.f
```

```
## [1] brown green brown blue  blue  blue

## Levels: blue brown green
```

Factor in R

- Level of factor:

```
levels(eye.col.f)  # The levels (distinct values) of the factor (categorical var)
```

```
## [1] "blue"  "brown" "green"
```

- Convert factor into vector:

```
as.numeric(eye.col.f)  # As numeric values: 1 is  blue, 2 is brown, 3 is green
```

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

```
as.numeric(eye.col.v)  # The character vector can not be coerced to numeric
```

```
## Warning: NAs introduced by coercion
```

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

```
as.character(eye.col.f)
```

```
## [1] "brown" "green" "brown" "blue"  "blue"  "blue"
```

Matrix in R

- Define matrix:

```r
m <- rep(1, 20)   # A vector of 20 elements, all 1

dim(m) <- c(5,4)  # Dimensions set to 5 & 4, so m is now a 5x4 matrix
```

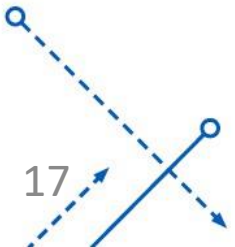
Creating a matrix using `matrix()`:

```r
m <- matrix(data=1, nrow=5, ncol=4)  # same matrix as above, 5x4, full of 1s

m <- matrix(1,5,4)                       # same matrix as above

dim(m)                                # What are the dimensions of m?
```

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

Creating a matrix by combining vectors:

```r
m <- cbind(1:5, 5:1, 5:9)  # Bind 3 vectors as columns, 5x3 matrix

m <- rbind(1:5, 5:1, 5:9)  # Bind 3 vectors as rows, 3x5 matrix
```



Matrix in R

- Access element of matrix:

```r
m <- matrix(1:10,10,10)



m[2,3]  # Matrix m, row 2, column 3 - a single cell

m[2,]   # The whole second row of m as a vector

m[,2]   # The whole second column of m as a vector

m[1:2,4:6] # submatrix: rows 1 and 2, columns 4, 5 and 6

m[-1,]     # all rows *except* the first one
```

Matrix in R

- Compare matrix:

```
# Are elements in row 1 equivalent to corresponding elements from column 1:

m[1,]==m[,1]

# A logical matrix: TRUE for m elements >3, FALSE otherwise:

m>3

# Selects only TRUE elements - that is ones greater than 3:

m[m>3]
```

Matrix in R

- Transformation operators in matrix:

```
t(m)            # Transpose m

m %*% t(m)      # %*% does matrix multiplication

m * m           # * does element-wise multiplication
```

Multidimensional array

- Array with more than 2 dimensions: use array()

```
a <- array(data=1:18,dim=c(3,3,2)) # 3d with dimensions 3x3x2

a <- array(1:18,c(3,3,2))          # the same array
```

List in R

- List in R is a set of objects.
  - Indexes can be numbers or names defined.
  - Contain elements of different types like numbers, string, vector, matrix, …

```
l1 <- list(boo=v1,foo=v2,moo=v3,zoo="Animals!")  # A list with four components

l2 <- list(v1,v2,v3,"Animals!")
```

```
l3 <- list()

l4 <- NULL
```

List in R

- Access elements in list:

```
l1["boo"]   # Access boo with single brackets: this returns a list.

l1[["boo"]] # Access boo with double brackets: this returns the numeric vector

l1[[1]]     # Returns the first component of the list, equivalent to above.

l1$boo      # Named elements can be accessed with the $ operator, as with [[]]
```

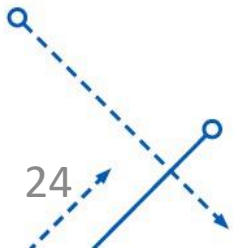
List in R

- Manipulating with elements in list:

```
l3[[1]] <- 11 # add an element to the empty list l3

l4[[3]] <- c(22, 23) # add a vector as element 3 in the empty list l4.
```

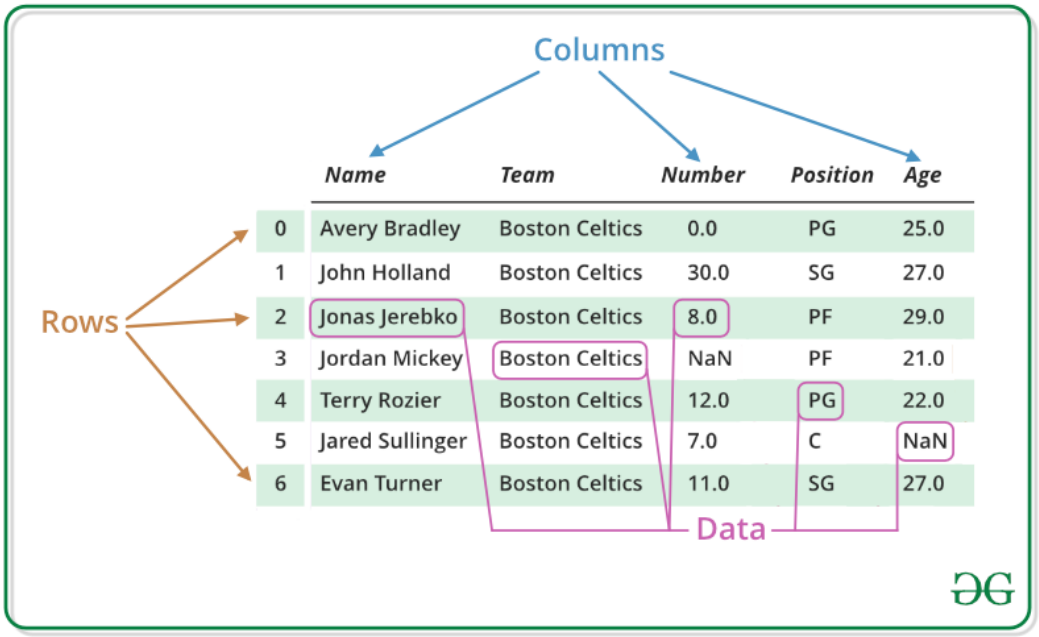
```
l1[[5]] <- "More elements!" # The list l1 had 4 elements, we're adding a 5th here.

l1[[8]] <- 1:11
```

```
l1$Something <- "A thing"  # Adds a ninth element - "A thing", named "Something"
```



Dataframe in R

- Dataframe is a special type of list to store data as a table.
  - Each row is a data sample
  - Each column is a feature (vector or factor)



Dataframe in R

- Create dataframe:

```
dfr1 <- data.frame( ID=1:4,

                    FirstName=c("John","Jim","Jane","Jill"),

                    Female=c(F,F,T,T),

                    Age=c(22,33,44,55) )



dfr1$FirstName   # Access the second column of dfr1.
```

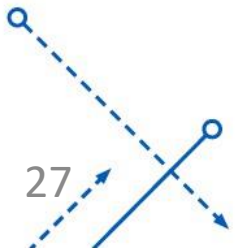
```
## [1] John Jim  Jane Jill

## Levels: Jane Jill Jim John
```

Dataframe in R

- Create columns as vectors in dataframe:

```
dfr1$FirstName <- as.vector(dfr1$FirstName)
```

```
dfr2 <- data.frame(FirstName=c("John","Jim","Jane","Jill"), stringsAsFactors=F)

dfr2$FirstName   # Success: not a factor.
```



Dataframe in R

- Access data in dataframe:

```
dfr1[1,]   # First row, all columns

dfr1[,1]   # First column, all rows

dfr1$Age   # Age column, all rows

dfr1[1:2,3:4] # Rows 1 and 2, columns 3 and 4 - the gender and age of John & Jim

dfr1[c(1,3),] # Rows 1 and 3, all columns
```

Dataframe in R

- Manipulations in dataframe:
  - Find the name of people with age > 30

```
dfr1[dfr1$Age>30,2]
```

```
## [1] "Jim"  "Jane" "Jill"
```

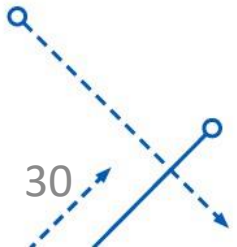
  - Find the average age of female:

```
mean ( dfr1[dfr1$Female==TRUE,4] )
```

```
## [1] 49.5
```

Content

- **Introduction to R language and its syntaxes**
- **Vector, Factor, Matrix, List, DataFrame in R**
- **Plot visualization with R**
- Create and generate graph
- Read and save graph data
- Visualization with igraph
- Describe features of graph



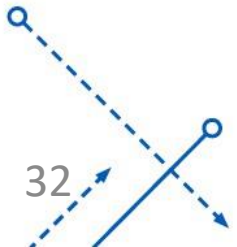
Plotting

- plot() is a basic tool for drawing in R.
- Consider the following data:

```
> head(mtcars)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1

> str(mtcars)
'data.frame':  32 obs. of  11 variables:
$ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
$ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
$ disp: num  160 160 108 258 360 ...
$ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
$ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
$ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
$ qsec: num  16.5 17 18.6 19.4 17 ...
$ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
$ am  : num  1 1 1 0 0 0 0 0 0 0 ...
$ gear: num  4 4 4 3 3 3 3 4 4 4 ...
$ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

Input for plot()

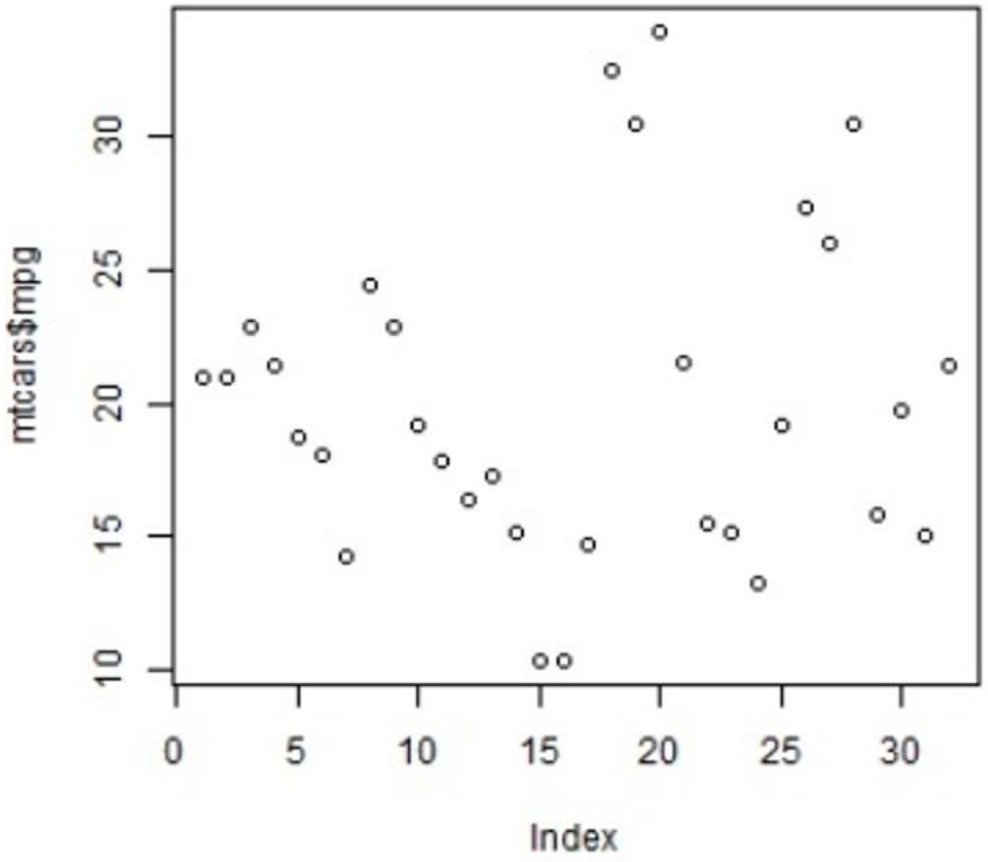
- plot() can describe these types of data:
  - Continuous variable
  - Discrete variable
  - 2 continuous variables
  - 2 discrete variable
  - 1 continuous and 1 discrete variable
  - 1 discrete and 1 continuous variable



A continuous variable

```
# plot a single continuous variable
plot(mtcars$mpg)
```

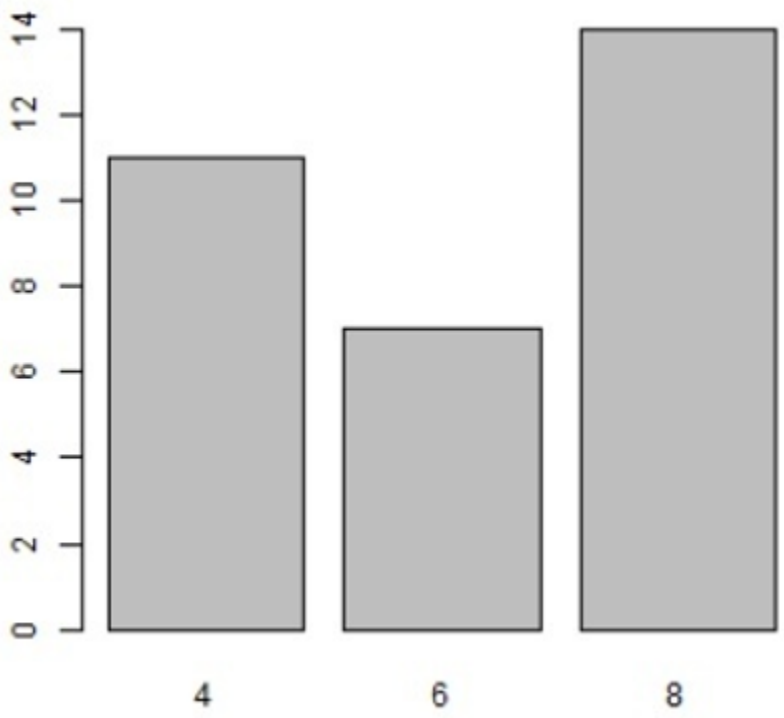
- plot() would create Scatter plot.



A discrete variable

```
# plot a single categorical variable
plot(mtcars$cyl)
```

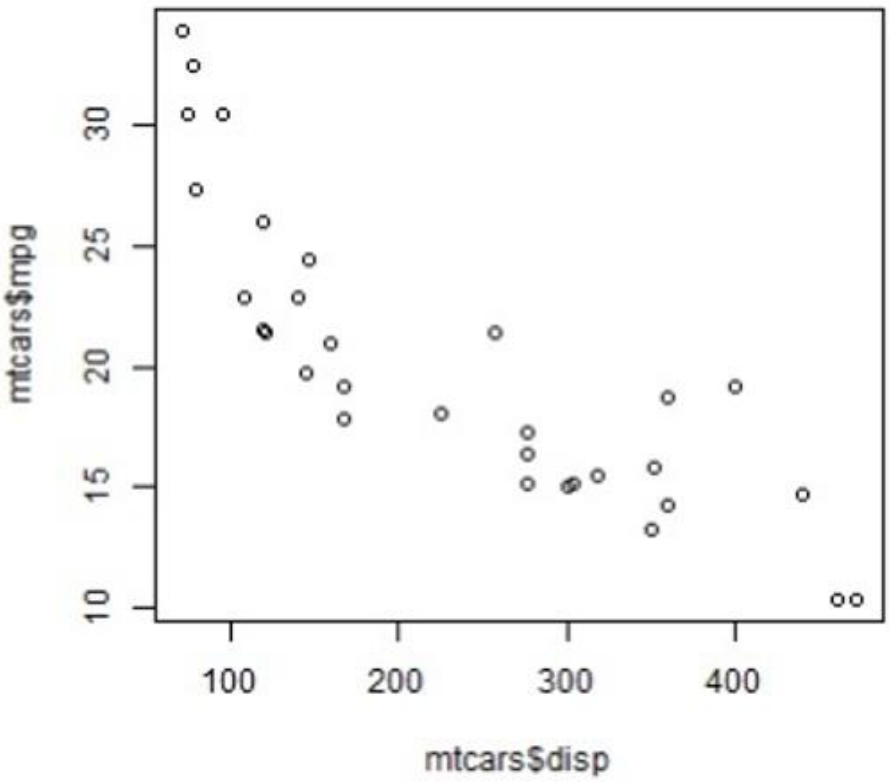
- plot() would create bar plot.



2 continuous variables

```
# plot two continuous variables
plot(mtcars$disp, mtcars$mpg)
```

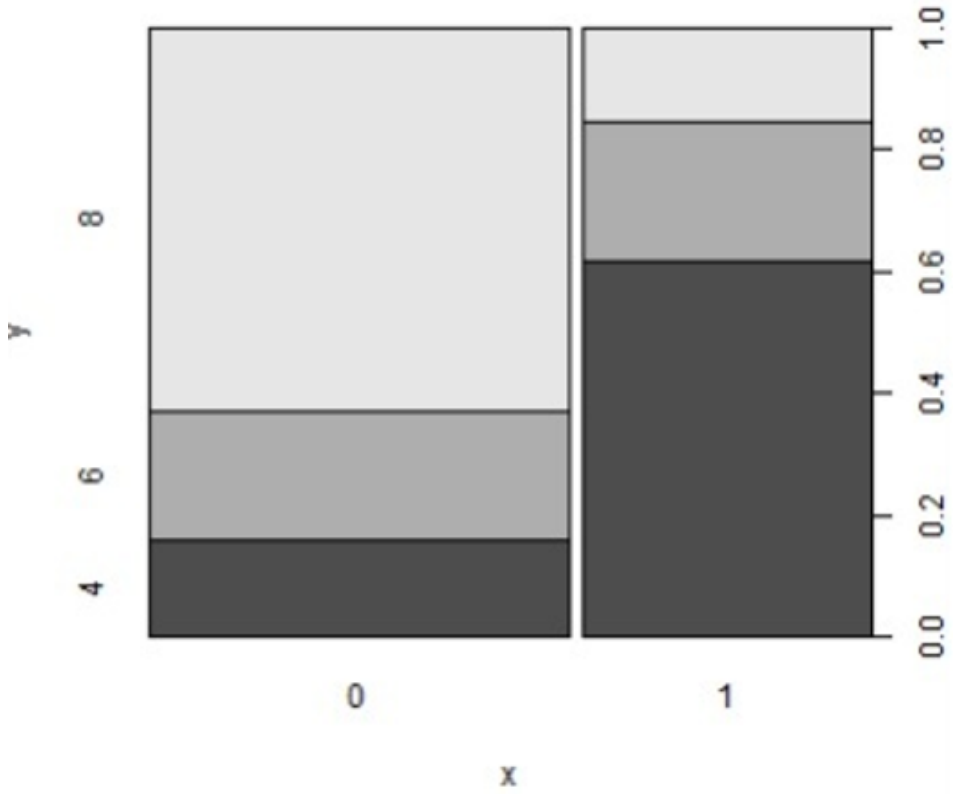
- plot() would create scatter plot which describes the correlation between 2 variables.



2 discrete variables

```
# plot two categorical variables
plot(mtcars$am, mtcars$cyl)
```

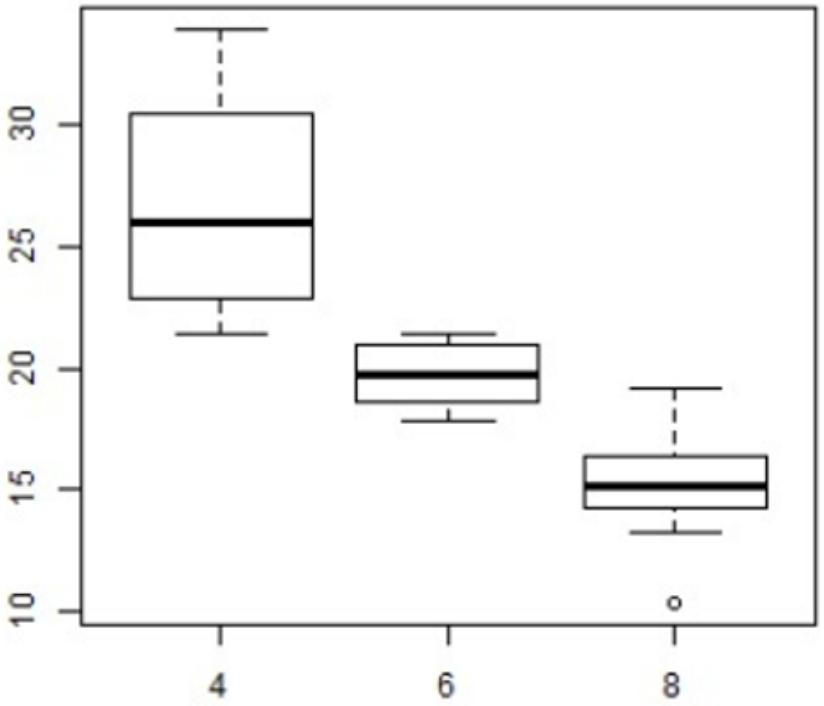
- plot() would create stacked bar plot.



A discreate and a continuous variable

```
# categorical/continuous variables
plot(mtcars$cyl, mtcars$mpg)
```

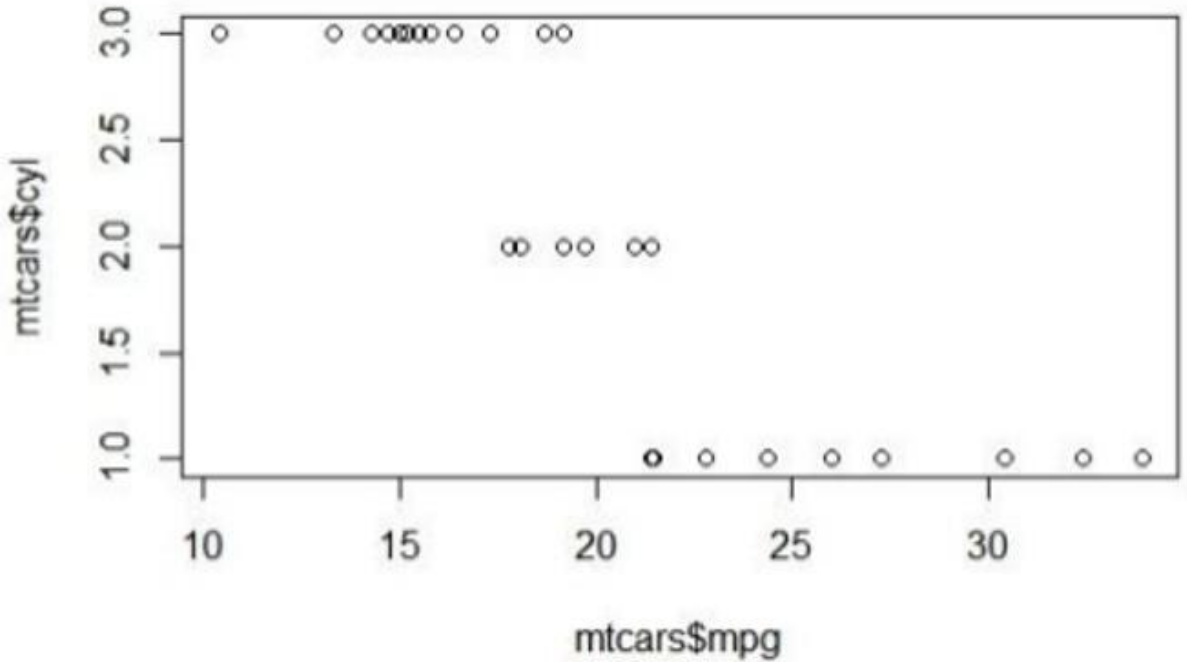
- Plot() would create box plot to illustrate the relationship between discrete and continuous variable.



A continuous and a discrete variable

```
# continuous vs categorical variables
plot(mtcars$mpg, mtcars$cyl)
```

- In the case that continuous variable is the first argument (X axis) and discreate variable is the second argument (Y axis), Scatter plot would be used.



Describe plot information

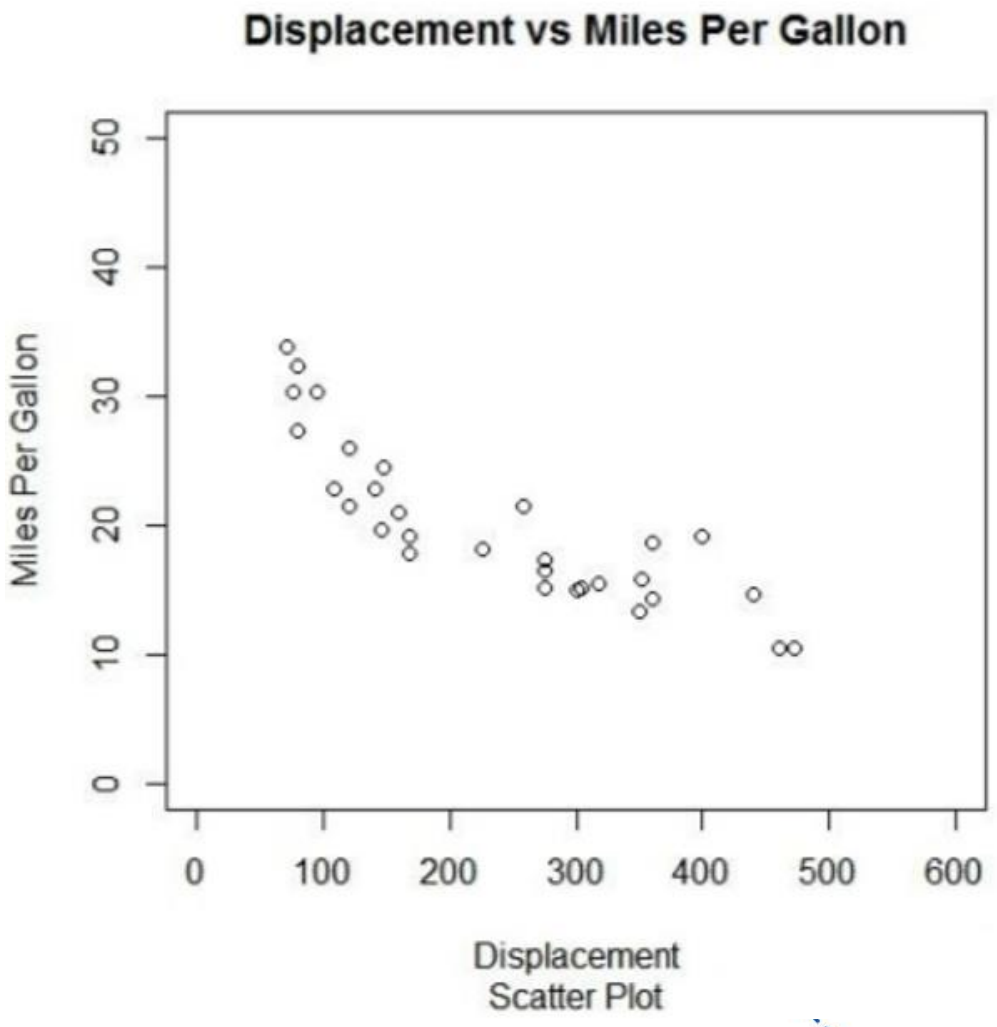
- The plot function also has arguments for the user to describe the diagram being plotted.

Feature	Argument	Value	Example
Title	`main`	String	"Scatter Plot"
Subtitle	`sub`	String	"Displacement vs Miles Per Gallon"
X Axis Label	`xlab`	String	"Displacement"
Y Axis Label	`ylab`	String	"Miles Per Gallon"
X Axis Range	`xlim`	Numeric Vector	c(0, 500)
Y Axis Range	`ylim`	Numeric Vector	c(0, 50)

Describe plot information

- Example:

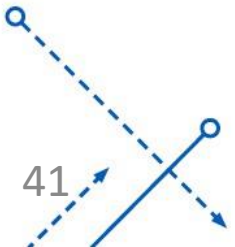
```
# create a plot with title, subtitle,  axis
labels and range
plot(mtcars$disp, mtcars$mpg,
    main = "Displacement vs Miles Per Gallon",
    sub = "Scatter Plot",
    xlab = "Displacement",
    ylab = "Miles Per Gallon",
    xlim = c(0, 600), ylim = c(0, 50))
```



Coloring

- col is an argument which is used to change the color for plot.

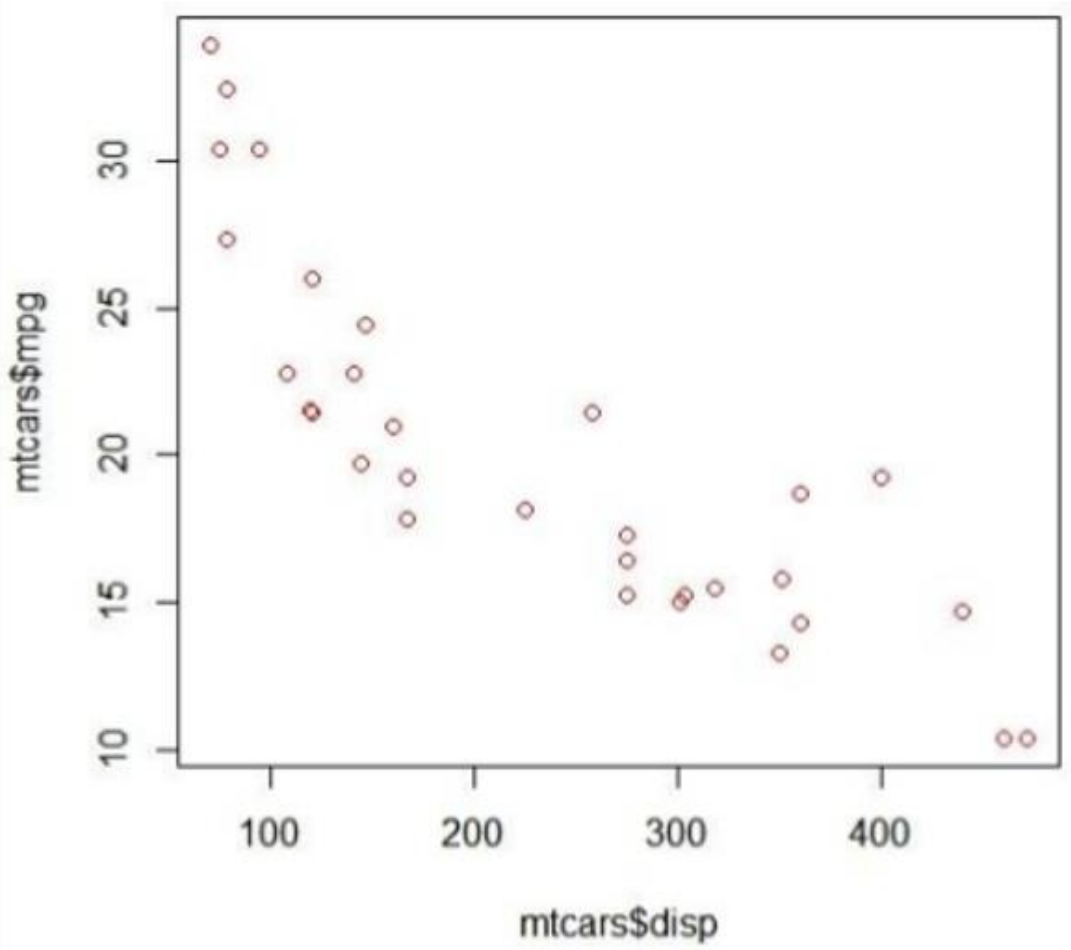
Feature	Argument	Value	Example
Symbol	`col`	String Hexadecimal RGB	"blue"
Title	`col.main`		"#0000ff"
Subtitle	`col.sub`		rgb(0, 0, 1)
Axis	`col.axis`		"red"
Label	`col.lab`		"#ff0000"
Foreground	`fg`		rgb(1, 0, 0)



Coloring

- Example:

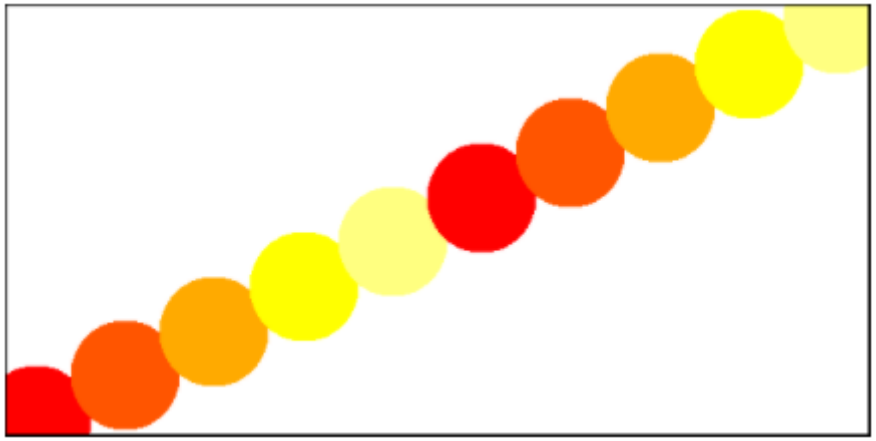
```
# modify color of the plot
plot(mtcars$disp, mtcars$mpg,
     col= "red")

                          OR

plot(mtcars$disp, mtcars$mpg,
     col = "#ff0000")

                          OR

plot(mtcars$disp, mtcars$mpg,
     col = rgb(1, 0, 0))
```



Coloring

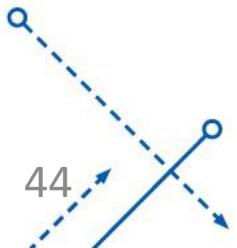
- We can use various colors for the same plots.

```
pal1 <- heat.colors(5, alpha=1)   #  5 colors from the heat palette, opaque
plot(x=1:10, y=1:10, col=pal1)
```



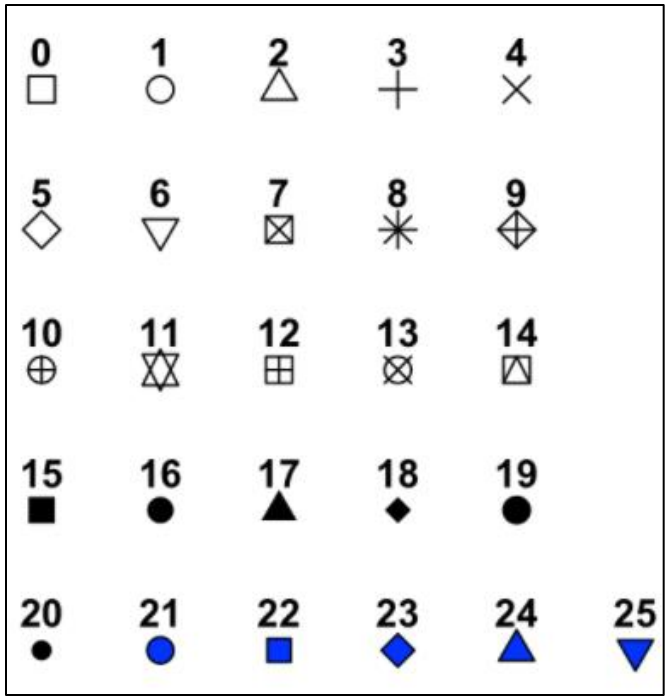
Point Shaping for Scatter

- For graph visualization, Scatter plot is usually in used.
- Attributes for representing a point in plot:
  - pch: points' shape
  - col: points' colors (symbol)
  - bg: background
  - cex: points' size
  - lwd: line thickness



Pch parameters

- R supports 26 different types of point shapes with a range from 0 to 25.

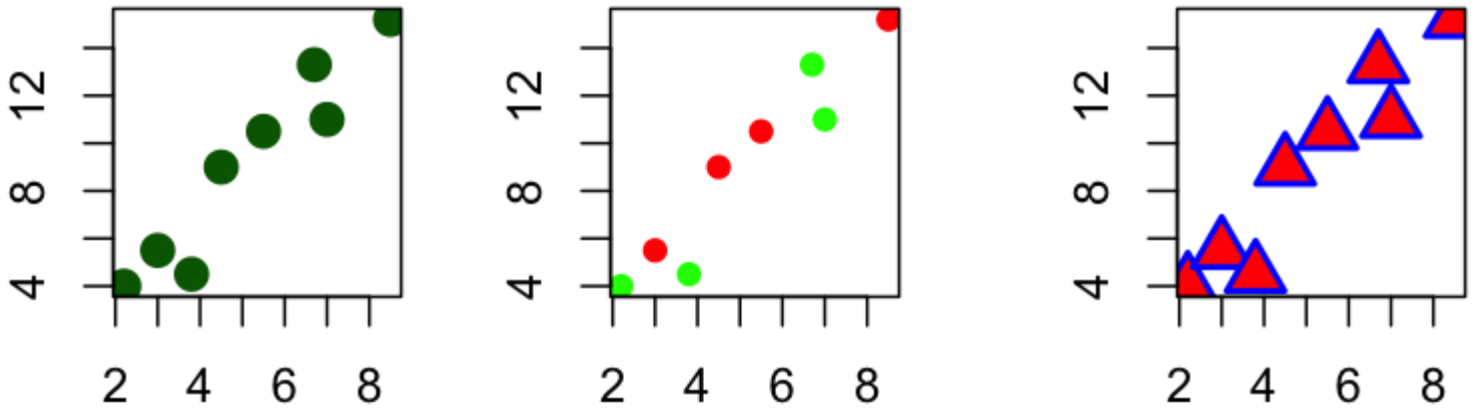


- pch = 0,square
- pch = 1,circle
- pch = 2,triangle point up
- pch = 3,plus
- pch = 4,cross
- pch = 5,diamond
- pch = 6,triangle point down
- pch = 7,square cross
- pch = 8,star
- pch = 9,diamond plus
- pch = 10,circle plus
- pch = 11,triangles up and down
- pch = 12,square plus
- pch = 13,circle cross
- pch = 14,square and triangle down
- pch = 15, filled square
- pch = 16, filled circle
- pch = 17, filled triangle point-up
- pch = 18, filled diamond
- pch = 19, solid circle
- pch = 20,bullet (smaller circle)
- pch = 21, filled circle blue
- pch = 22, filled square blue
- pch = 23, filled diamond blue
- pch = 24, filled triangle point-up blue
- pch = 25, filled triangle point down blue

Point Shaping for Scatter

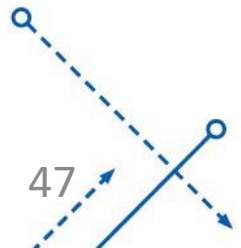
- Examples:

```
# Change color
plot(x, y, pch=19, col="darkgreen", cex=1.5)
# Color can be a vector
plot(x, y, pch=19, col=c("green", "red"))
# change border, background color and line width
plot(x, y, pch = 24, cex=2, col="blue", bg="red", lwd=2)
```



Content

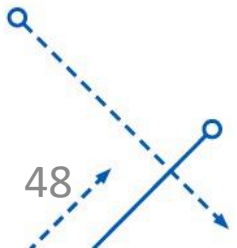
- **Introduction to R language and its syntaxes**
- **Vector, Factor, Matrix, List, DataFrame in R**
- **Plot visualization with R**
- **Create and generate graph**
- Read and save graph data
- Visualization with igraph
- Describe features of graph



Igraph library

- In order to deal with graph in R, a library developed quite well is igraph.
- Installing igraph in R as follow:

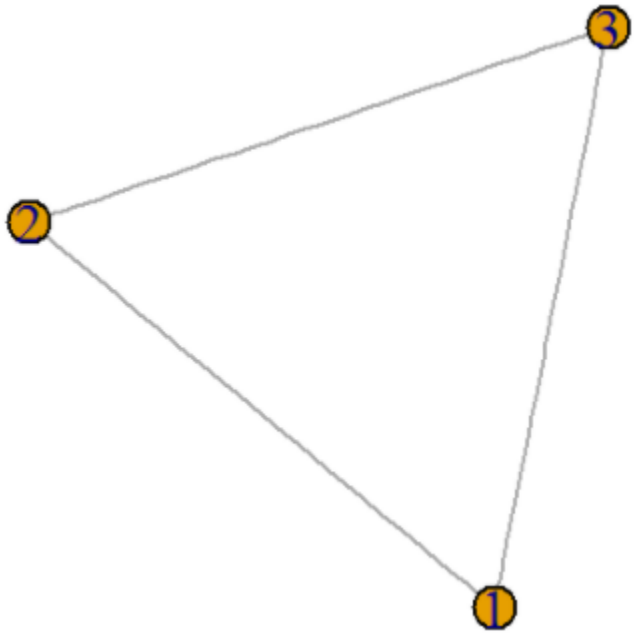
```
## Download and install the package
install.packages("igraph")

## Load package
library(igraph)
```



Create graph

- Create an undirected graph with 3 edges:

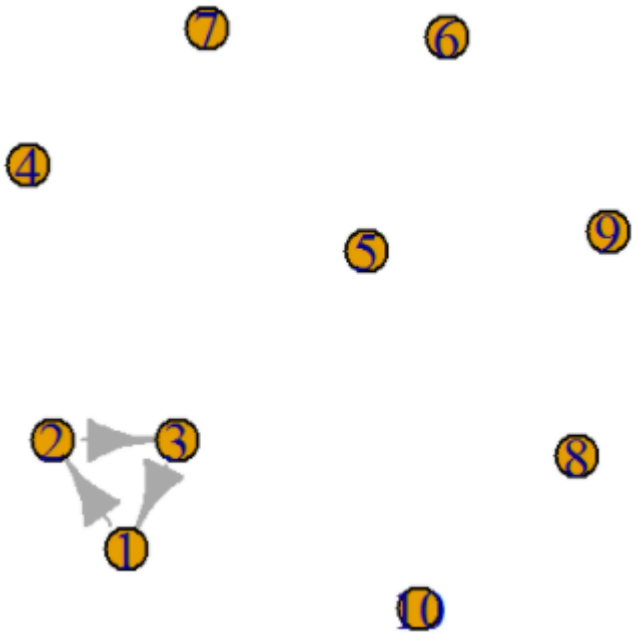
```
g1 <- graph( edges=c(1,2, 2,3, 3, 1), n=3, directed=F )

plot(g1) # A simple plot of the network
```



Create graph

- Create a directed graph with 10 nodes and 3 edges:

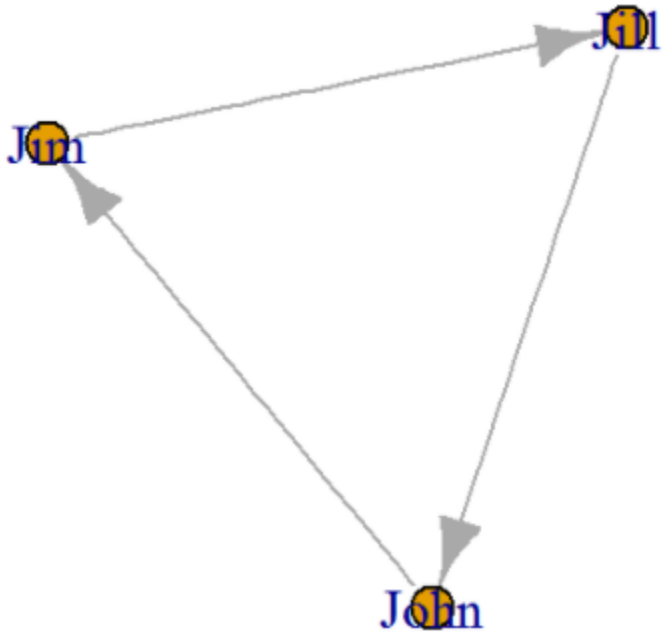
```
# Now with 10 vertices, and directed by default:

g2 <- graph( edges=c(1,2, 2,3, 3, 1), n=10 )

plot(g2)
```



Create graph

- Create graph with label nodes:

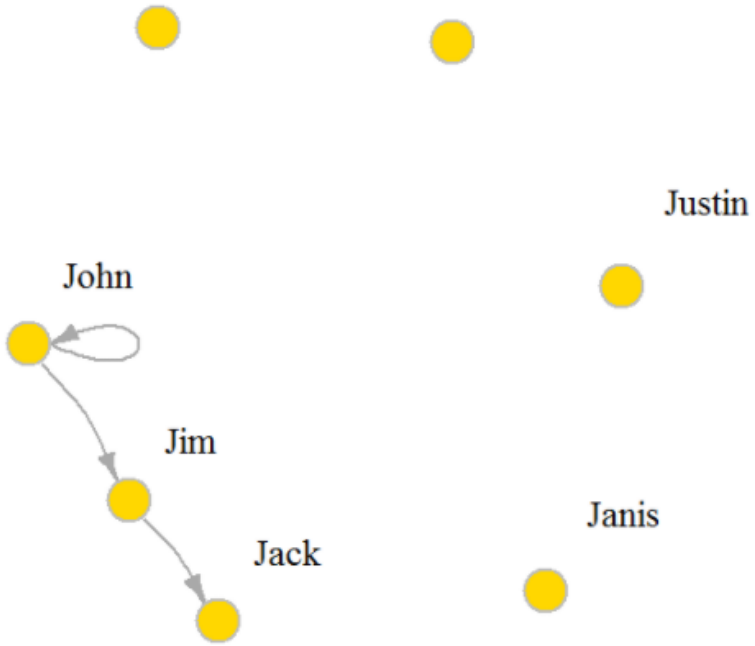
```
g3 <- graph( c("John", "Jim", "Jim", "Jill", "Jill", "John")) # named vertices

# When the edge list has vertex names, the number of nodes is not needed

plot(g3)
```



Create graph

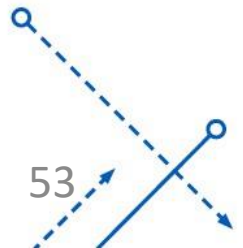
- Create graph with isolated nodes:

```
g4 <- graph( c("John", "Jim", "Jim", "Jack", "Jim", "Jack", "John", "John"),
             isolates=c("Jesse", "Janis", "Jennifer", "Justin") )
# In named graphs we can specify isolates by providing a list of their names.
```



Create graph with symbols

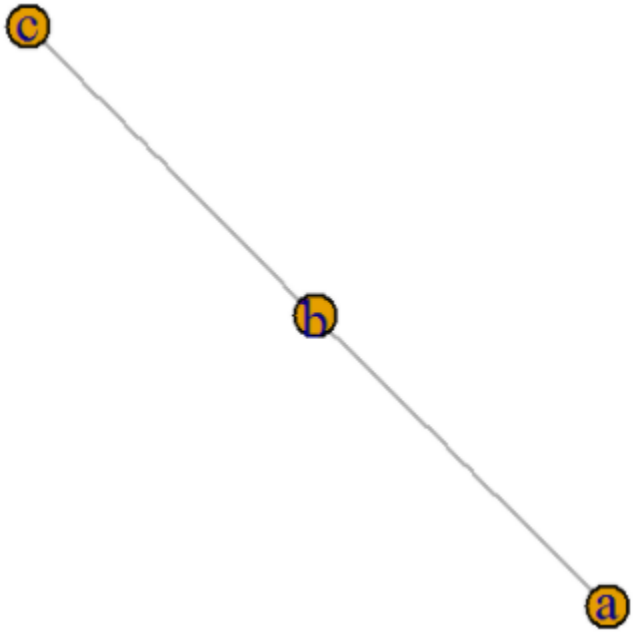
- Create graph simply with symbols:
  - Sign – represents undirected edge
  - Sign +- or -+ represent directed edge (left or right)
  - Sign ++ represents for both directions.
  - Sign : represents for nodes' list.



Create graph with symbols

- Example 1:

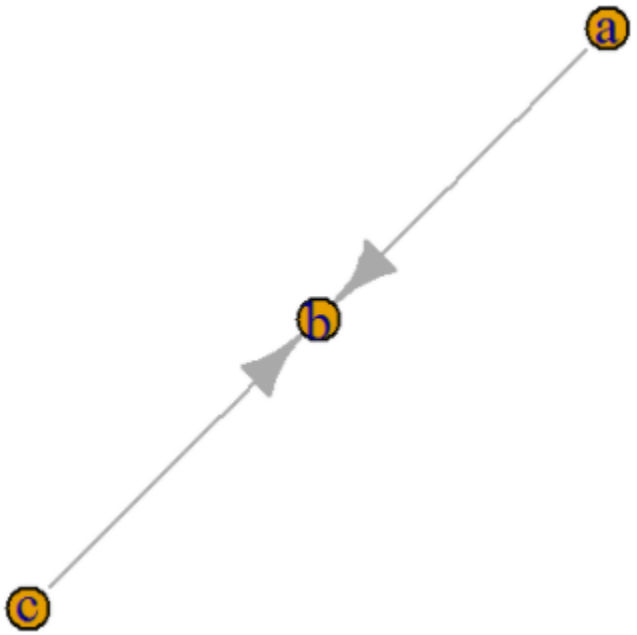
```
plot(graph_from_literal(a---b, b---c)) # the number of dashes doesn't matter
```



Create graph with symbols

- Example 2:

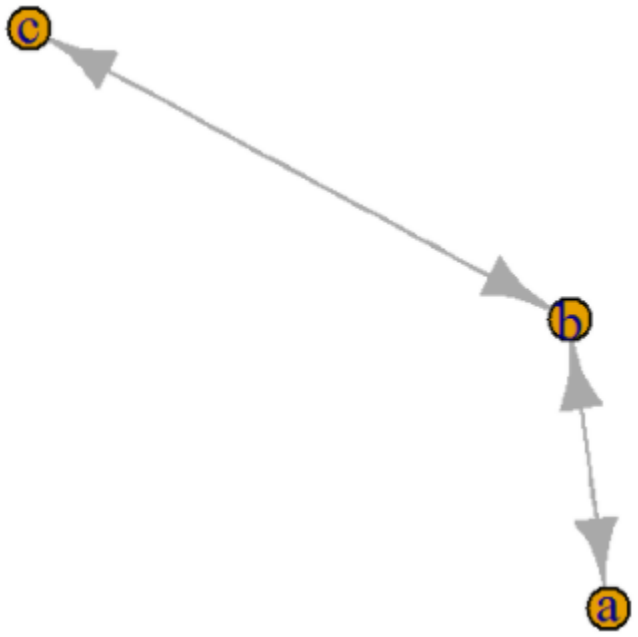
```
plot(graph_from_literal(a--+b, b+--c))
```



Create graph with symbols

- Example 3:

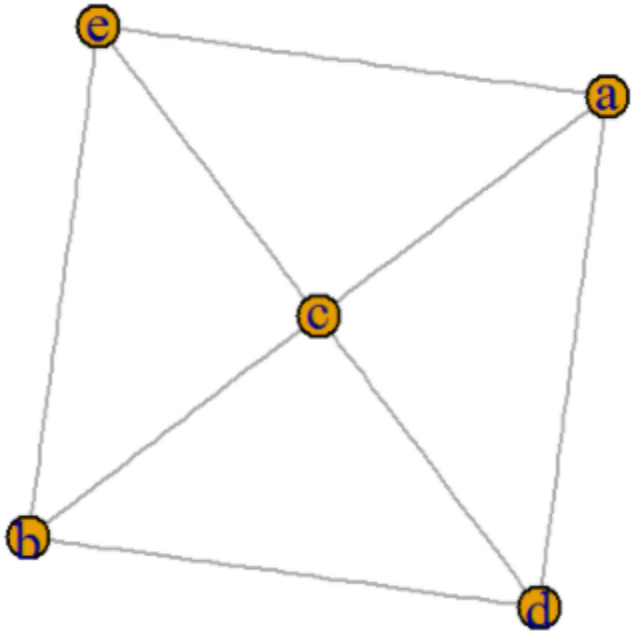
```
plot(graph_from_literal(a+-+b, b+-+c))
```



Create graph with symbols

- Example 4:

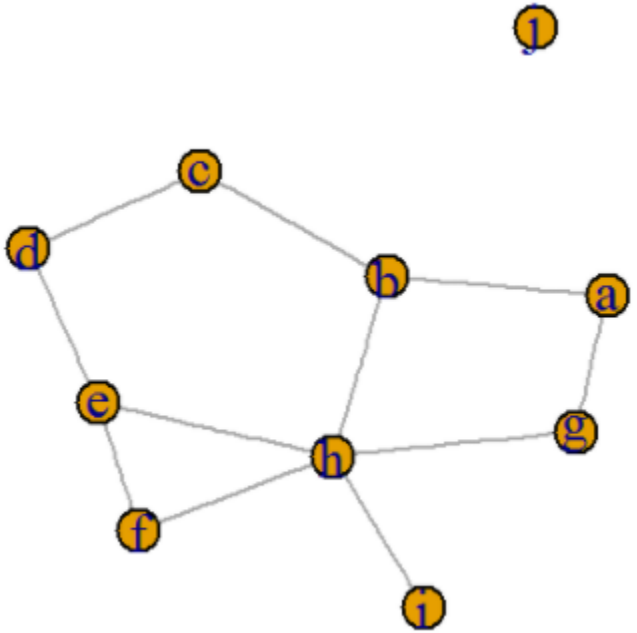
```
plot(graph_from_literal(a:b:c---c:d:e))
```



Create graph with symbols

- Example 5:

```
gl <- graph_from_literal(a-b-c-d-e-f, a-g-h-b, h-e:f:i, j)

plot(gl)
```



Graph features

- Edges:

```
E(g4) # The edges of the object
```

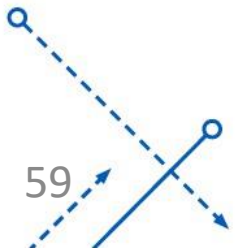
- Nodes:

```
V(g4) # The vertices of the object
```

- Matrix representing graph:

```
g4[]
```

```
g4[1,]
```

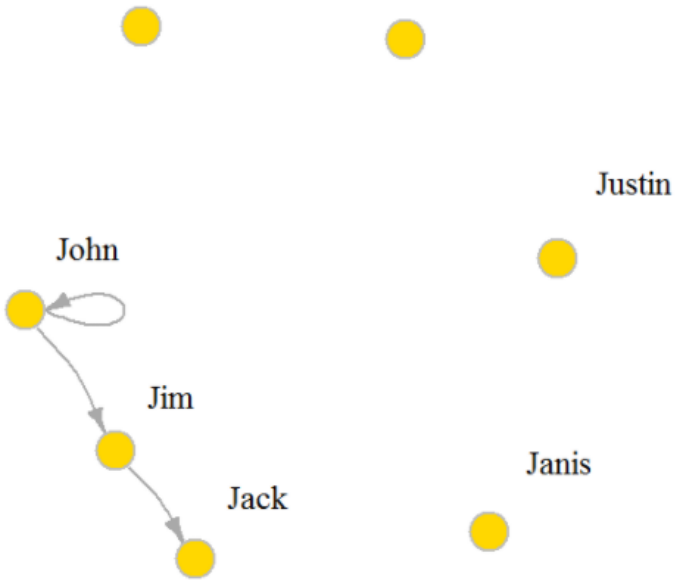


Graph features

- $name$ is a default attribute to store node' name.

```
V(g4)$name # automatically generated when we created the network.
```

```
## [1] "John"     "Jim"      "Jack"     "Jesse"    "Janis"    "Jennifer"

## [7] "Justin"
```

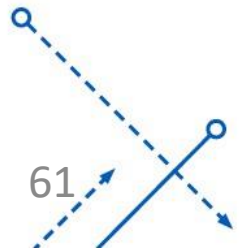


Graph features

- When viewing a graph, the igraph library displays a summary of information about the graph.

```
g4s
```

```
## IGRAPH DNW- 7 3 -- Email Network

## + attr: name (g/c), name (v/c), gender (v/c), weight (e/n)

## + edges (vertex names):

## [1] John->John John->Jim  Jim ->Jack
```



Graph features

- Information includes: Graph type, number of nodes, number of edges, features, node list, …
  - Graph type:
    - D/U: directed (D) or undirected (U)
    - N: A graph containing vertices with the attribute "name."
    - W: A graph containing edges with the attribute "weight."
    - B: A bipartite graph (vertices with the attribute "type").
  - Mô tả thuộc tính đỉnh, cạnh:
    - (g/c): thuộc tính ký tự (c) mức đồ thị (g)
    - (v/c): thuộc tính ký tự (c) mức đỉnh (v)
    - (e/n): thuộc tính số (n) mức cạnh

```
g4s
```

```
## IGRAPH DNW- 7 3 -- Email Network

## + attr: name (g/c), name (v/c), gender (v/c), weight (e/n)

## + edges (vertex names):

## [1] John->John John->Jim  Jim ->Jack
```

Adding features for graph

- Adding features for nodes, edges in graph:

```
V(g4)$gender <- c("male", "male", "male", "male", "female", "female", "male")

E(g4)$type <- "email" # Edge attribute, assign "email" to all edges

E(g4)$weight <- 10    # Edge weight, setting all existing edges to 10
```

- Display features:

```
edge_attr(g4)
```

```
vertex_attr(g4)
```

- On the other hand , igraph also supports functions set_edge_attr(), set_vertex_attr() to change nodes and edges' features.

Adding features for graph

- Adding features for graph:

```
g4 <- set_graph_attr(g4, "name", "Email Network")

g4 <- set_graph_attr(g4, "something", "A thing")
```

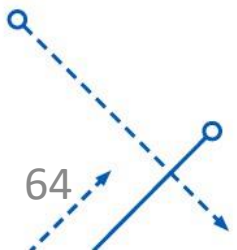
- Display feature for graph:

```
graph_attr(g4, "name")
```

```
graph_attr(g4)
```

- Remove graph's feature:

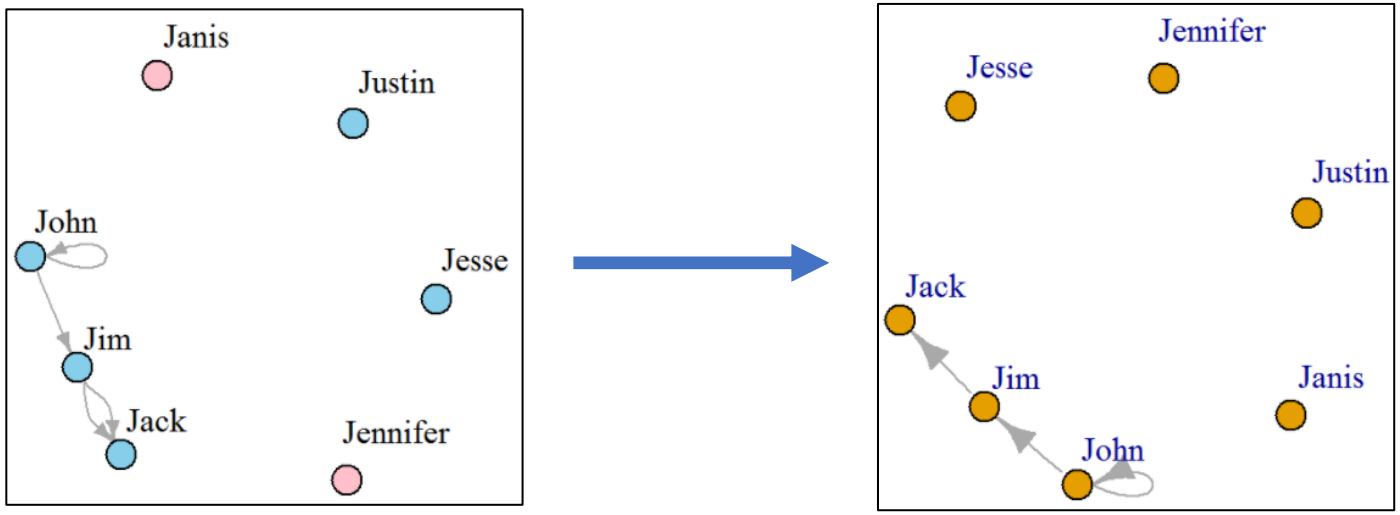
```
g4 <- delete_graph_attr(g4, "something")
```



Simplizing graph

- We can simplizing the graph by removing self loops and multiple edges.
  - edge.attr.comb describes how features to be combined

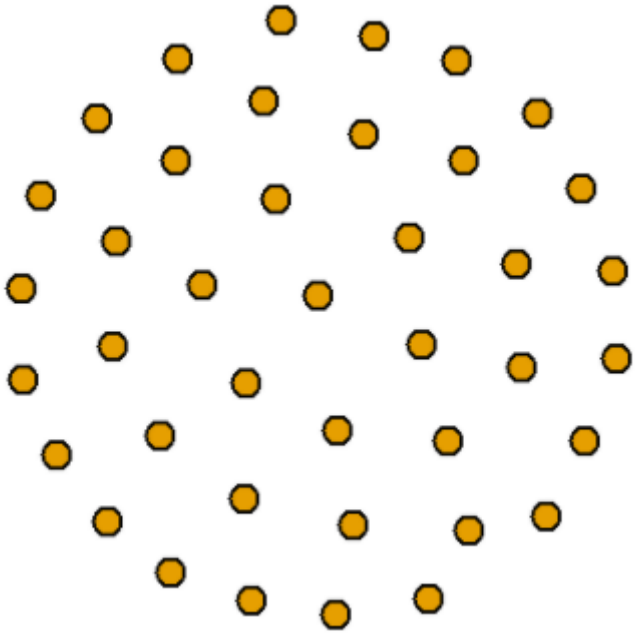
```
g4s <- simplify( g4, remove.multiple = T, remove.loops = F,
                     edge.attr.comb=c(weight="sum", type="ignore") )
```



Generating Graph

- Generate empty graph:

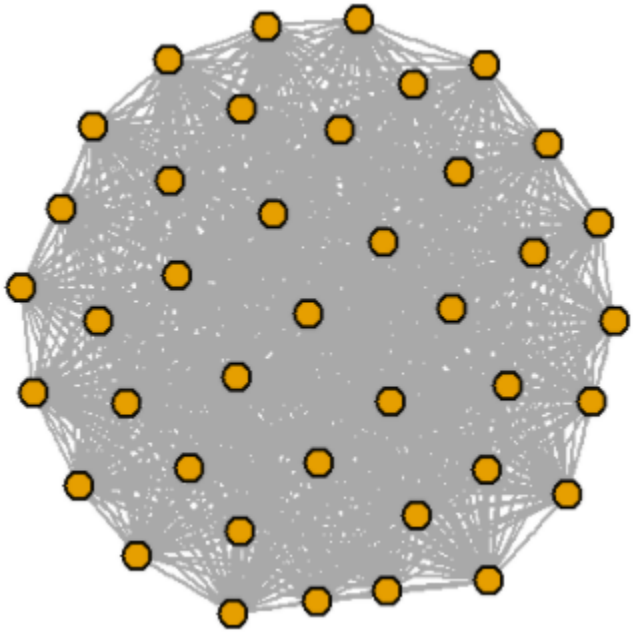
```
eg <- make_empty_graph(40)

plot(eg, vertex.size=10, vertex.label=NA)
```



Generating Graph

- Generate complete graph:

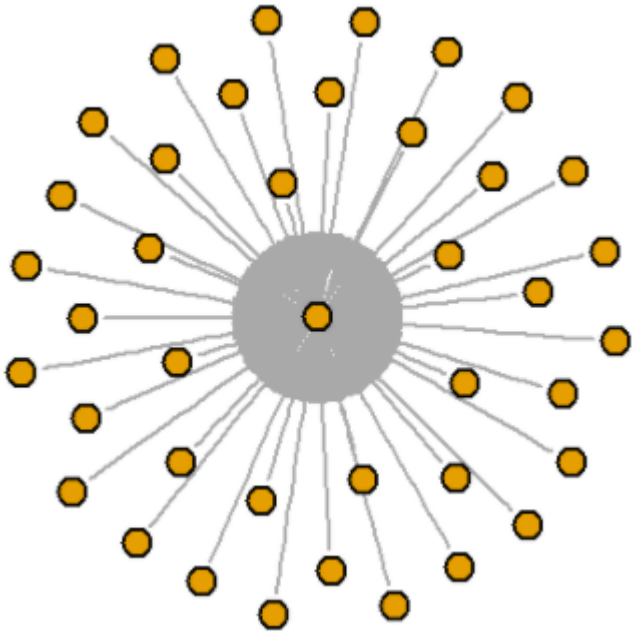
```
fg <- make_full_graph(40)

plot(fg, vertex.size=10, vertex.label=NA)
```



Generating Graph

- Generate star graph:

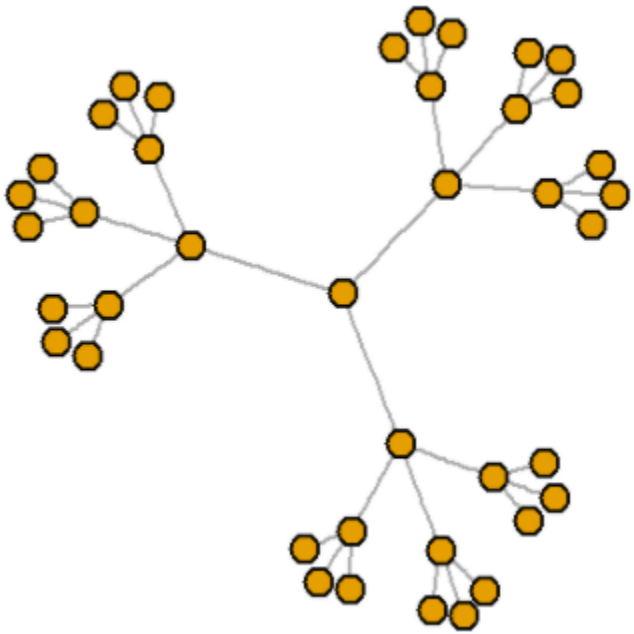
```
st <- make_star(40)

plot(st, vertex.size=10, vertex.label=NA)
```



Generating Graph

- Generate tree graph:

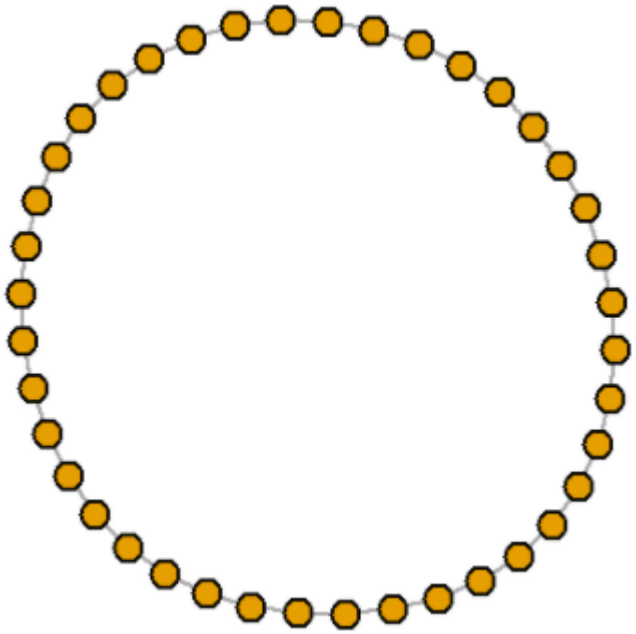
```
tr <- make_tree(40, children = 3, mode = "undirected")

plot(tr, vertex.size=10, vertex.label=NA)
```



Generating Graph

- Generate circular graph:

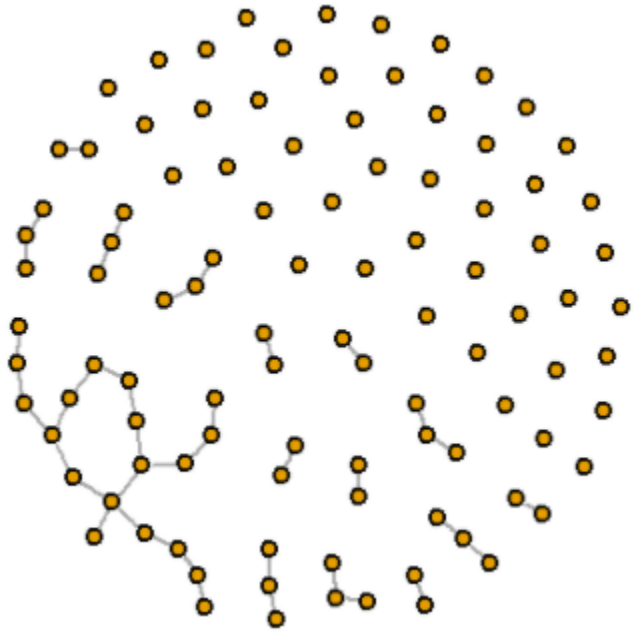
```
rn <- make_ring(40)

plot(rn, vertex.size=10, vertex.label=NA)
```



Generating Graph

- Generate Erdos-Renyi model:

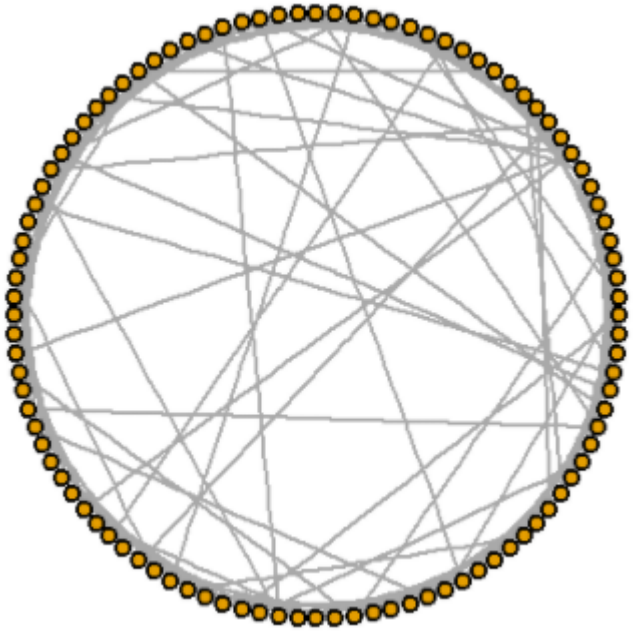
```
er <- sample_gnm(n=100, m=40)

plot(er, vertex.size=6, vertex.label=NA)
```



Generating Graph

- Generate small world Watts-Strogatz:

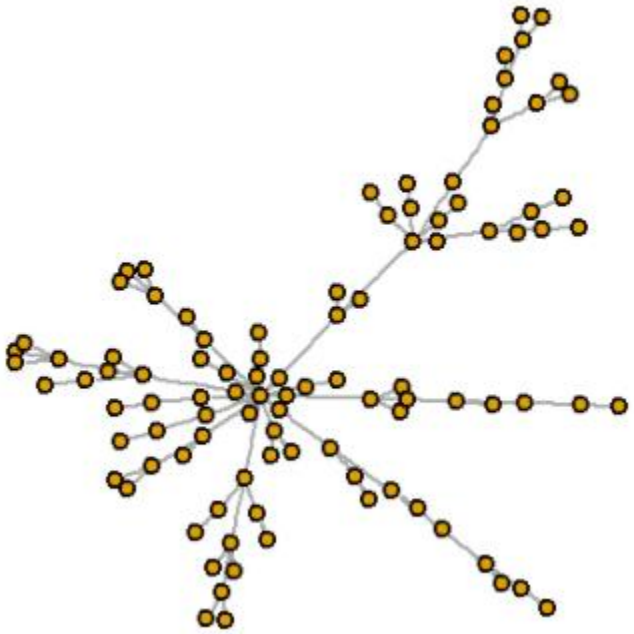
```
sw <- sample_smallworld(dim=2, size=10, nei=1, p=0.1)

plot(sw, vertex.size=6, vertex.label=NA, layout=layout_in_circle)
```



Generating Graph

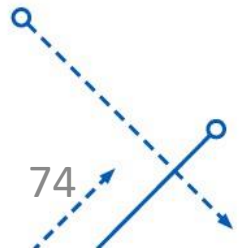
- Generate Barabasi-Albert model:

```
ba <-  sample_pa(n=100, power=1, m=1,  directed=F)

plot(ba, vertex.size=6, vertex.label=NA)
```



Content

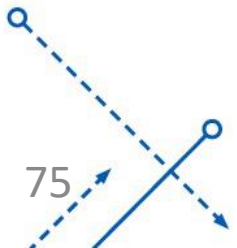
- **Introduction to R language and its syntaxes**
- **Vector, Factor, Matrix, List, DataFrame in R**
- **Plot visualization with R**
- **Create and generate graph**
- **Read and save graph data**
- Visualization with igraph
- Describe features of graph



Read graph data from data sources

- The igraph library has the capability to read graph data from various data sources
- Reading from file:

```
dat=read.csv(file.choose(),header=TRUE,row.names=1,check.names=FALSE)
m=as.matrix(dat)
net=graph.adjacency(m,mode="directed",weighted=TRUE,diag=FALSE)
```



Read graph data from data sources

- igraph can read graph data from various data sources.
- Read from http connection:

```
advice_data_frame <- read.table('http://sna.stanford.edu/sna_R_labs/data/Krack-High-Tec-edgelist-Advice.txt')
g <- graph.data.frame(advice_data_frame)
```

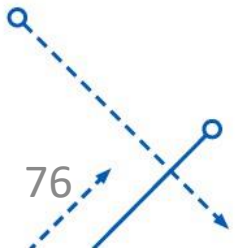
- Other file types:

Load graph by edges:

```
g <- read.graph("./graph.txt", format="edgelist")
```

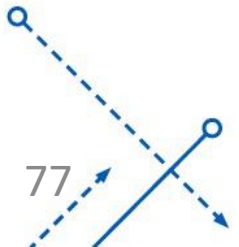
Load graph in pajek format:

```
g <- read.graph("./graph.dl", format="pajek")
```



Convert data into graph by igraph

- We can convert raw data into graph by using graph.data.frame with 2 arguments:
  - d: describe edge of graph. Hai cột đầu là id của đỉnh nguồn và đích, các cột sau là các thuộc tính của cạnh
  - vertices: describe vertices. Cột đầu là id của đỉnh, các cột tiếp theo là thuộc tính của đỉnh.



Convert data into graph by igraph

- For example, we have edges and vertices list in file:

```
from,to,weight,type
s01,s02,10,hyperlink
s01,s02,12,hyperlink
s01,s03,22,hyperlink
s01,s04,21,hyperlink
s04,s11,22,mention
s05,s15,21,mention
s06,s17,21,mention
s08,s09,11,mention
s08,s09,12,mention
s03,s04,22,hyperlink
s04,s03,23,hyperlink
```

```
id,media,media.type,type.label,audience.size
s01,NY Times,1,Newspaper,20
s02,Washington Post,1,Newspaper,25
s03,Wall Street Journal,1,Newspaper,30
s04,USA Today,1,Newspaper,32
s05,LA Times,1,Newspaper,20
s06,New York Post,1,Newspaper,50
s07,CNN,2,TV,56
s08,MSNBC,2,TV,34
s09,FOX News,2,TV,60
s10,ABC,2,TV,23
s11,BBC,2,TV,34
s12,Yahoo News,3,Online,33
```

- Read data:

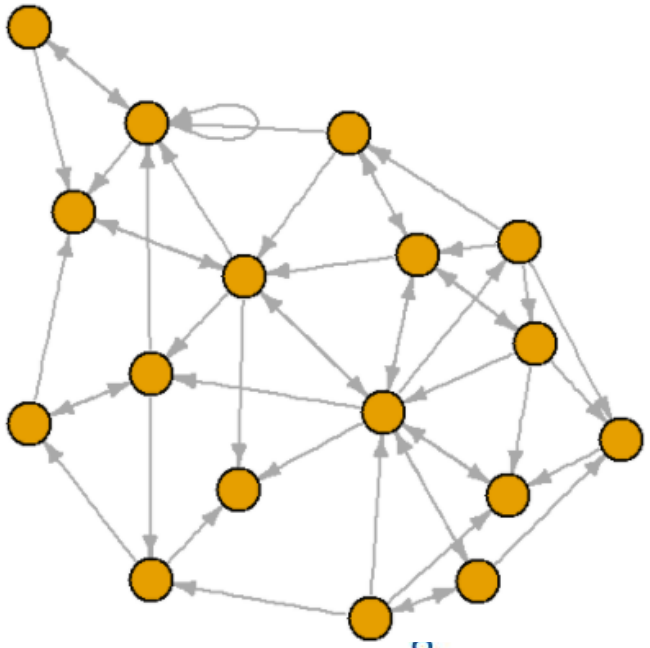
```
nodes <- read.csv("Dataset1-Media-Example-NODES.csv", header=T, as.is=T)

links <- read.csv("Dataset1-Media-Example-EDGES.csv", header=T, as.is=T)
```

Convert data into graph by igraph

- Convert into graph object in igraph:

```
net <- graph_from_data_frame(d=links, vertices=nodes, directed=T)
```

```
## IGRAPH DNW- 17 49 --
## + attr: name (v/c), media (v/c), media.type (v/n), type.label
## | (v/c), audience.size (v/n), type (e/c), weight (e/n)
## + edges (vertex names):
##  [1] s01->s02 s01->s03 s01->s04 s01->s15 s02->s01 s02->s03 s02->s09
##  [8] s02->s10 s03->s01 s03->s04 s03->s05 s03->s08 s03->s10 s03->s11
## [15] s03->s12 s04->s03 s04->s06 s04->s11 s04->s12 s04->s17 s05->s01
## [22] s05->s02 s05->s09 s05->s15 s06->s06 s06->s16 s06->s17 s07->s03
## [29] s07->s08 s07->s10 s07->s14 s08->s03 s08->s07 s08->s09 s09->s10
## [36] s10->s03 s12->s06 s12->s13 s12->s14 s13->s12 s13->s17 s14->s11
## [43] s14->s13 s15->s01 s15->s04 s15->s06 s16->s06 s16->s17 s17->s04
```



Convert data into graph by igraph

- Convert identity and adjacency matrix into graph object in igraph:

```
net2 <- graph_from_incidence_matrix(links2)
```

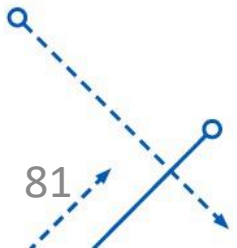
```
graph_from_adjacency_matrix()
```

```
,U01,U02,U03,U04,U05,U06,U07,U08,U09,U10,U11
s01,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
s02,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
s03,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0
s04,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0
s05,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0
s06,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,0,0,0,0
s07,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0
s08,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,0
s09,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1
s10,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0
```

Convert igraph data into raw data

- We can extract data from graph into edge list or matrix, or even data frame:

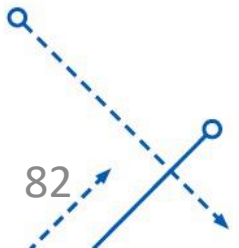
```
as_edgelist(net, names=T)

as_adjacency_matrix(net, attr="weight")

as_data_frame(net, what="edges")

as_data_frame(net, what="vertices")
```



Write graph data to file

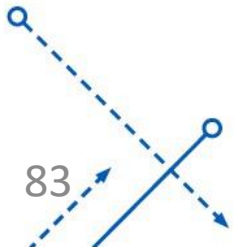
- Export data to file:

```
write.graph(g, file='my_graph.dl', format="pajek")
write.graph(g, file='my_graph.txt', format="edgelist")
```



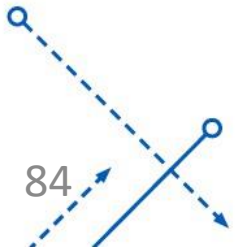
Content

- **Introduction to R language and its syntaxes**
- **Vector, Factor, Matrix, List, DataFrame in R**
- **Plot visualization with R**
- **Create and generate graph**
- **Read and save graph data**
- **Visualization with igraph**
- Describe features of graph



Plotting with igraph

- igraph also develop various plotting methods as well as graph-specific features than default plot in R.
- These features are expressed through vertex and edge arguments.



Plotting with igraph

- Vertex parameters:

vertex.color	Node color
vertex.frame.color	Node border color
vertex.shape	One of “none”, “circle”, “square”, “csquare”, “rectangle” “crectangle”, “vrectangle”, “pie”, “raster”, or “sphere”
vertex.size	Size of the node (default is 15)
vertex.size2	The second size of the node (e.g. for a rectangle)
vertex.label	Character vector used to label the nodes
vertex.label.family	Font family of the label (e.g.“Times”, “Helvetica”)
vertex.label.font	Font: 1 plain, 2 bold, 3, italic, 4 bold italic, 5 symbol
vertex.label.cex	Font size (multiplication factor, device-dependent)
vertex.label.dist	Distance between the label and the vertex
vertex.label.degree	The position of the label in relation to the vertex, where 0 right, “pi” is left, “pi/2” is below, and “-pi/2” is above

Plotting with igraph

- Edge parameters:

EDGES	
edge.color	Edge color
edge.width	Edge width, defaults to 1
edge.arrow.size	Arrow size, defaults to 1
edge.arrow.width	Arrow width, defaults to 1
edge.lty	Line type, could be 0 or “blank”, 1 or “solid”, 2 or “dashed”, 3 or “dotted”, 4 or “dotdash”, 5 or “longdash”, 6 or “twodash”
edge.label	Character vector used to label edges
edge.label.family	Font family of the label (e.g.“Times”, “Helvetica”)
edge.label.font	Font: 1 plain, 2 bold, 3, italic, 4 bold italic, 5 symbol
edge.label.cex	Font size for edge labels
edge.curved	Edge curvature, range 0-1 (FALSE sets it to 0, TRUE to 0.5)
arrow.mode	Vector specifying whether edges should have arrows, possible values: 0 no arrow, 1 back, 2 forward, 3 both

Plotting with igraph

- Other parameters:

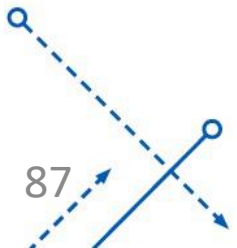
OTHER

margin Empty space margins around the plot, vector with length 4

frame if TRUE, the plot will be framed

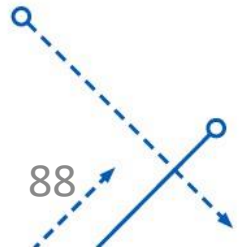
main If set, adds a title to the plot

sub If set, adds a subtitle to the plot



Plotting with igraph

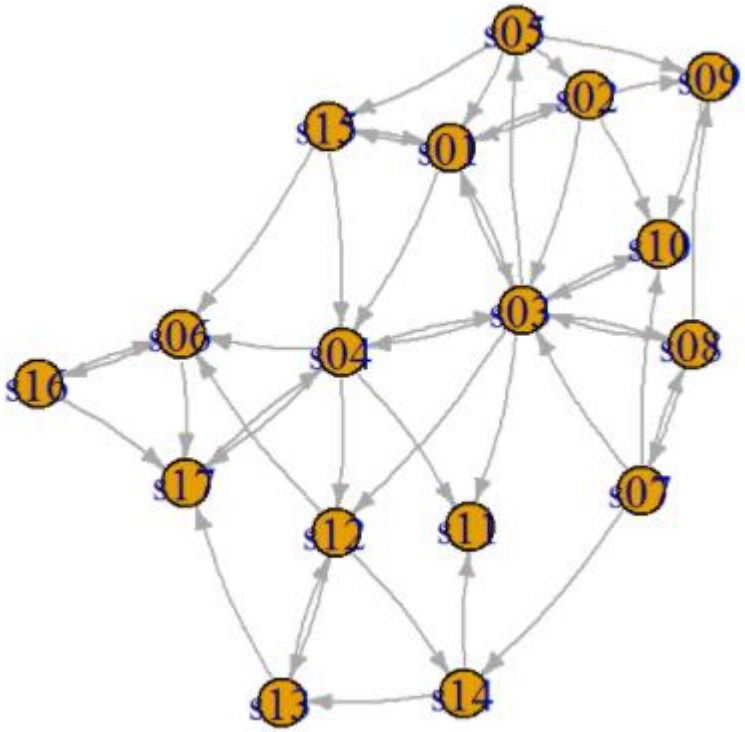
- There are 2 ways to configure plotting in igraph:
  - Configure through arguments in plot()
  - configure through igraph objects



Plotting with igraph

- Method 1: configure through arguments in plot()

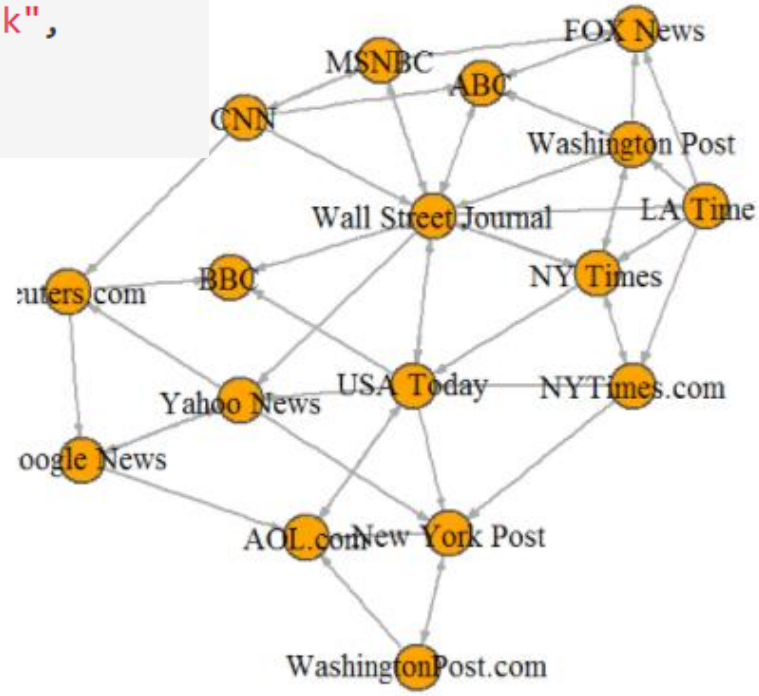
```
# Plot with curved edges (edge.curved=.1) and reduce arrow size:

plot(net, edge.arrow.size=.4, edge.curved=.1)
```



Plotting with igraph

- Method 1: configure through arguments in plot()

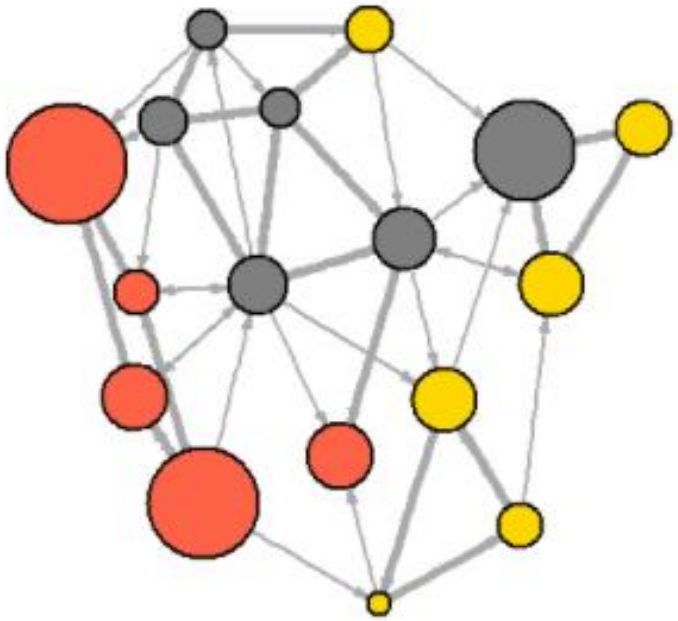
```
# Set edge color to gray, and the node color to orange.

# Replace the vertex label with the node names stored in "media"

plot(net, edge.arrow.size=.2, edge.curved=0,

     vertex.color="orange", vertex.frame.color="#555555",

     vertex.label=V(net)$media, vertex.label.color="black",

     vertex.label.cex=.7)
```



Plotting with igraph

- Method 2: configure through igraph objects.

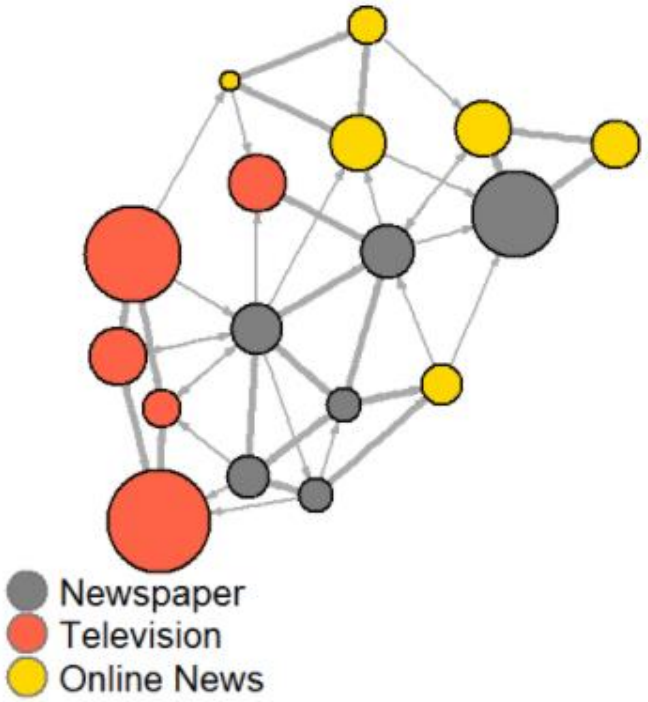
```
# Generate colors based on media type:
colrs <- c("gray50", "tomato", "gold")
V(net)$color <- colrs[V(net)$media.type]

# Set node size based on audience size:
V(net)$size <- V(net)$audience.size*0.7

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

# Set edge width based on weight:
E(net)$width <- E(net)$weight/6
#change arrow size and edge color:
E(net)$arrow.size <- .2
E(net)$edge.color <- "gray80"

E(net)$width <- 1+E(net)$weight/12
```



Plotting with igraph

- Adding labels for colors in graph

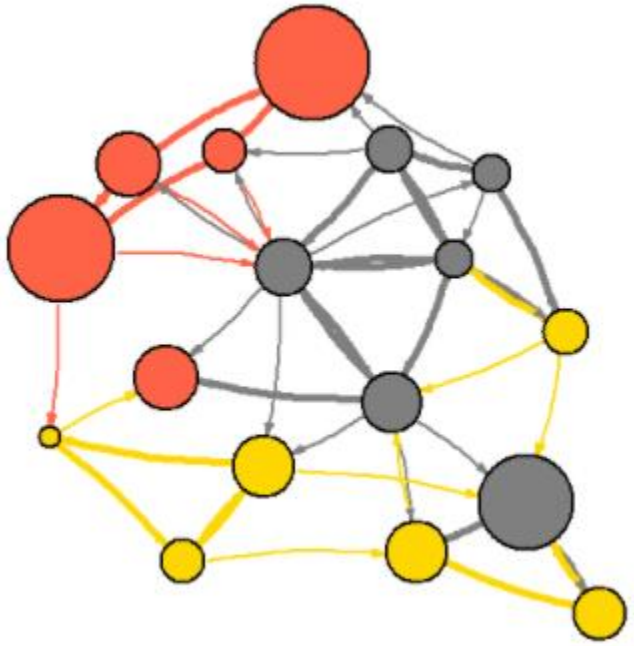
```
plot(net)

legend(x=-1.5, y=-1.1, c("Newspaper","Television", "Online News"), pch=21,

       col="#777777", pt.bg=colrs, pt.cex=2, cex=.8, bty="n", ncol=1)
```



Plotting with igraph

- Coloring edges

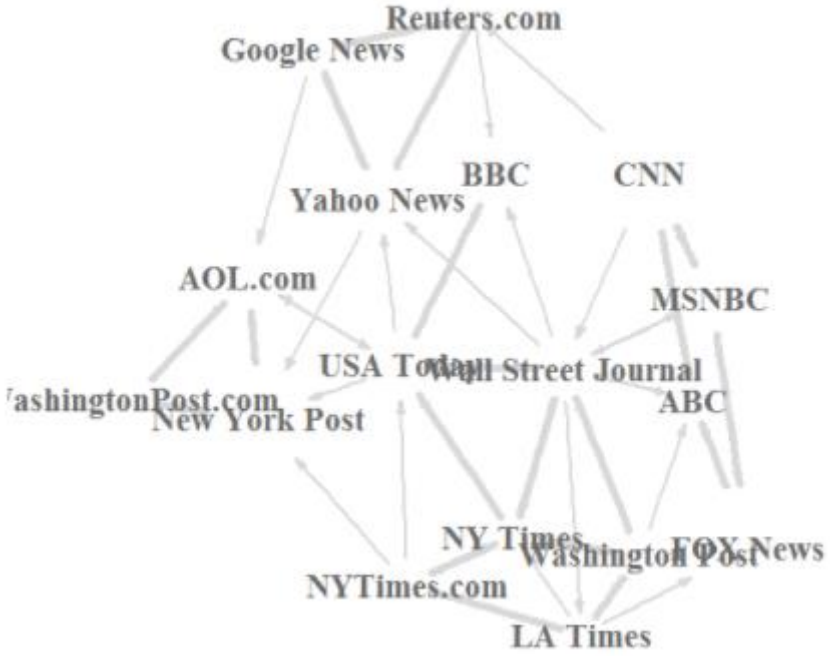
```
edge.start <- ends(net, es=E(net), names=F)[,1]

edge.col <- V(net)$color[edge.start]



plot(net, edge.color=edge.col, edge.curved=.1)
```



Plotting with igraph

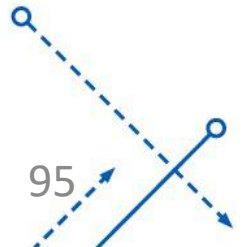
- In some graph, we just care about node labels so we can only display labels.

```
plot(net, vertex.shape="none", vertex.label=V(net)$media,
     vertex.label.font=2, vertex.label.color="gray40",
     vertex.label.cex=.7, edge.color="gray85")
```



Layouts

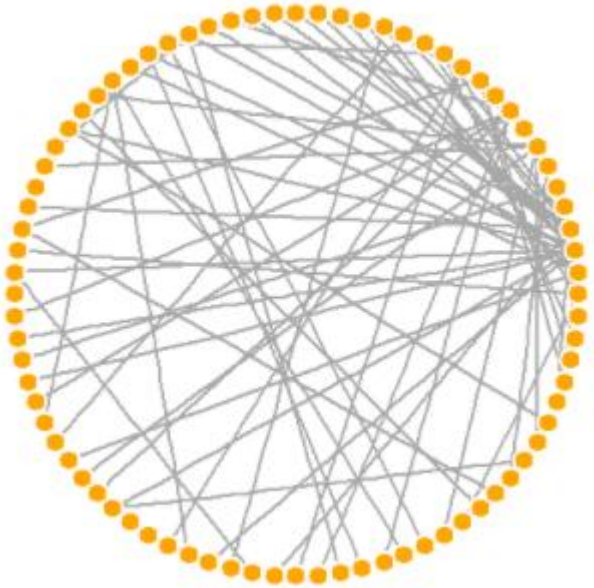
- Due to the large size of graph, we can visualize graph in various ways by using layout.
- There are 2 ways to configure layout:
  - Based on standard layout (built-in layout)
  - Based on coordinate list.



Layouts

- Example:

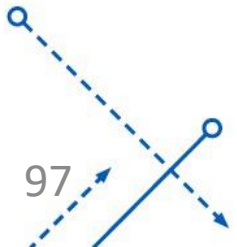
```
l <- layout_in_circle(net.bg)

plot(net.bg, layout=l)
```



```
# 3D sphere layout

l <- layout_on_sphere(net.bg)

plot(net.bg, layout=l)
```

Content

- **Introduction to R language and its syntaxes**
- **Vector, Factor, Matrix, List, DataFrame in R**
- **Plot visualization with R**
- **Create and generate graph**
- **Read and save graph data**
- **Visualization with igraph**
- **Describe features of graph**



Graph description

- R supports some statistic functions for graph.
- Edge density:

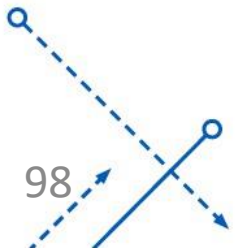
```
edge_density(net, loops=F)
```

- Diameter:

```
diameter(net, directed=F, weights=NA)
```

- Degree:

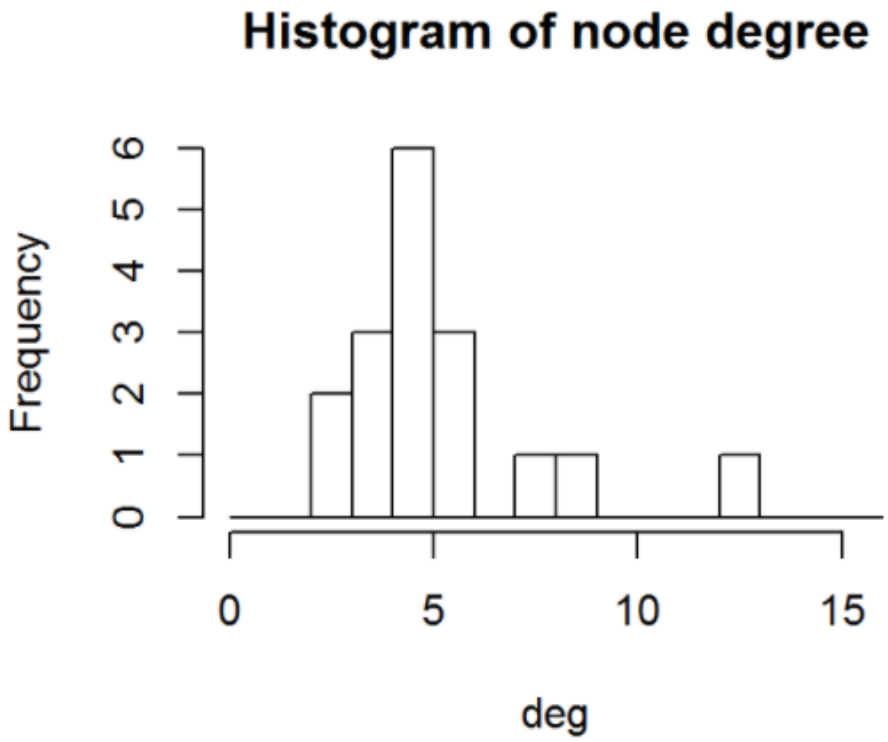
```
deg <- degree(net, mode="all")
```



Graph description

- Histogram of node degree:

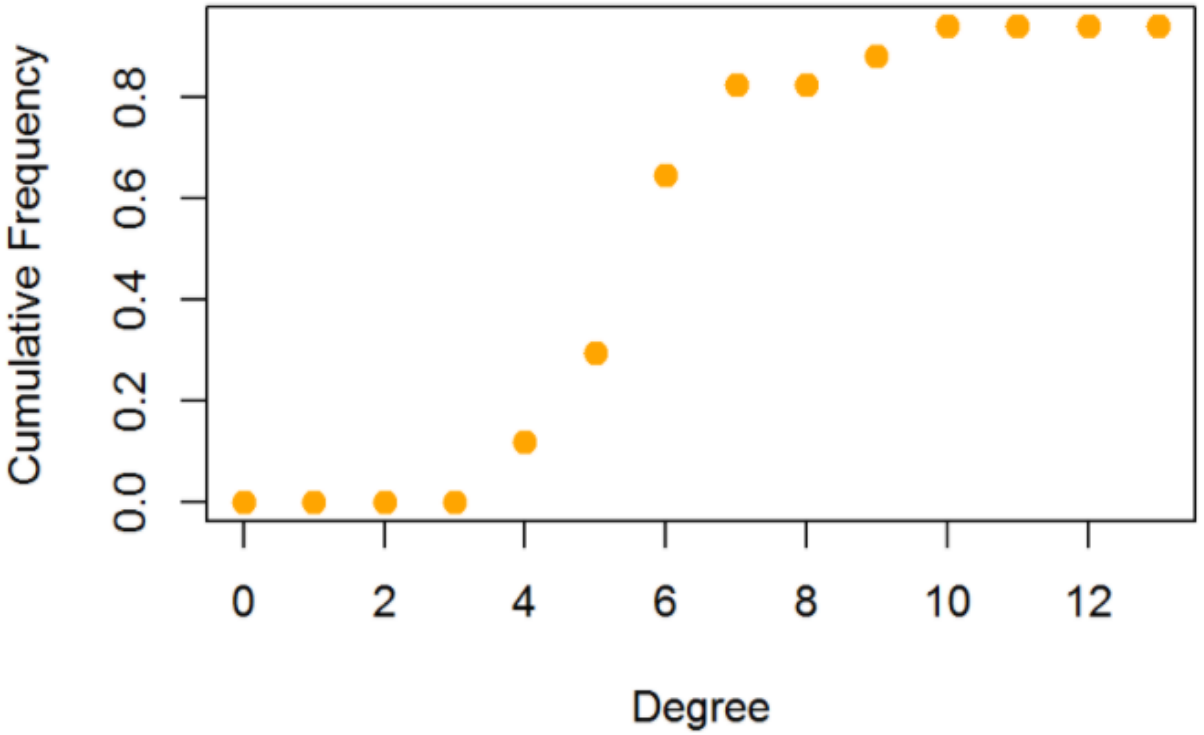
```
hist(deg, breaks=1:vcount(net)-1, main="Histogram of node degree")
```



Graph description

- Degree distribution:

```
deg.dist <- degree_distribution(net, cumulative=T, mode="all")

plot( x=0:max(deg), y=1-deg.dist, pch=19, cex=1.2, col="orange",

      xlab="Degree", ylab="Cumulative Frequency")
```



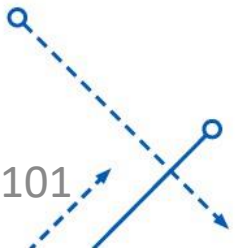
Graph description

- Centrality (degree, betweenness, closeness):

```
degree(net, mode="in")

centr_degree(net, mode="in", normalized=T)
```

```
closeness(net, mode="all", weights=NA)

centr_clo(net, mode="all", normalized=T)
```

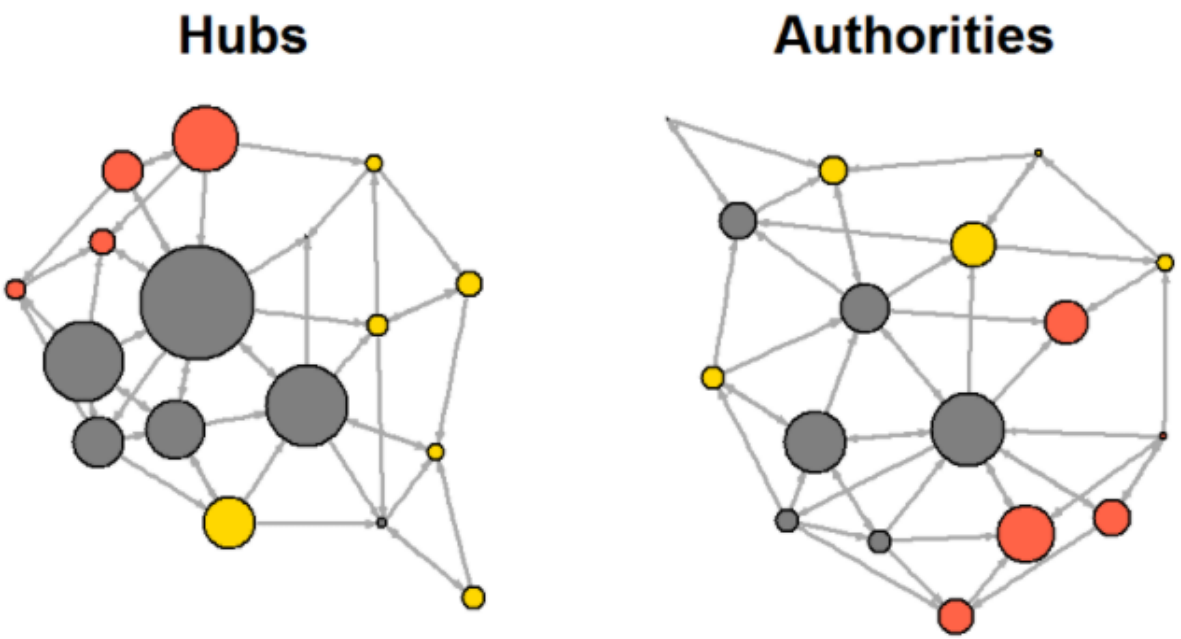
```
betweenness(net, directed=T, weights=NA)

edge_betweenness(net, directed=T, weights=NA)

centr_betw(net, directed=T, normalized=T)
```



Graph description

- Hub and Authority:

```
hs <- hub_score(net, weights=NA)$vector

as <- authority_score(net, weights=NA)$vector
par(mfrow=c(1,2))

 plot(net, vertex.size=hs*50, main="Hubs")

 plot(net, vertex.size=as*30, main="Authorities")
```



Graph description

- Distances, shortest paths, neighbors:

```
mean_distance(net, directed=F)
```

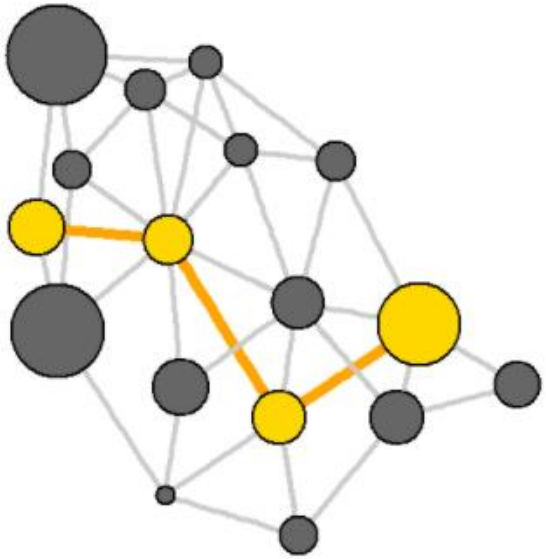
```
distances(net) # with edge weights

distances(net, weights=NA) # ignore weights
```

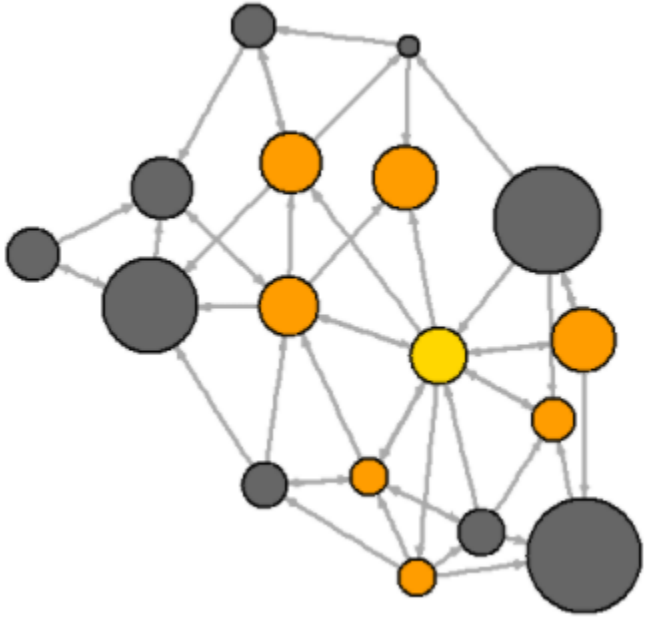
```
news.path <- shortest_paths(net,
                            from = V(net)[media=="MSNBC"],
                             to  = V(net)[media=="New York Post"],
                             output = "both") # both path nodes and edges
```

```
neigh.nodes <- neighbors(net, V(net)[media=="Wall Street Journal"], mode="out")
```

Graph description

- Distances, shortest paths, neighbors.

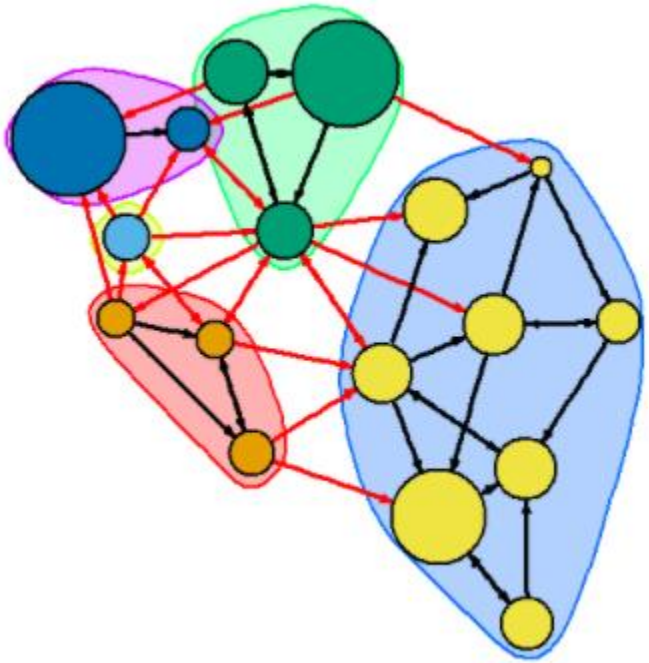




Graph description

- Community detection:

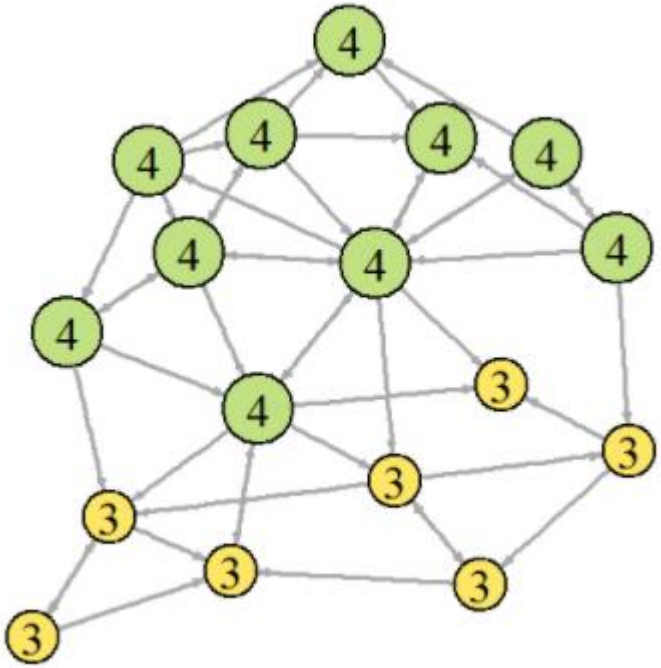
```
ceb <- cluster_edge_betweenness(net)

dendPlot(ceb, mode="hclust")
```



Graph description

- K-core decomposition:

```
kc <- coreness(net, mode="all")

plot(net, vertex.size=kc*6, vertex.label=kc, vertex.color=colrs[kc])
```



References

- https://kateto.net/netscix2016.html
- https://www.slideshare.net/RsquaredIn/r-data-visualization-for-beginners
- https://igraph.org/r/