Introduction to Network Analysis

Materials to Accompany Lecture 1: What is Network Analysis?

Valentina Kuskova, PhD

Contents

Welcome! **2**
Prerequisites . . . 2
Learning objectives . . . 3
Data . . . 3
Working with 'Materials' files . . . 3
Required packages . . . 3

What are networks? Random vs. non-random **4**
Creating a network object from the matrix data . . . 4
Generating a matrix of data . . . 4

A picture is worth a thousand words **6**
Importing network data in different formats . . . 6
Reading a Pajek file . . . 6
The classic: Hartford drug users . . . 7
Assignment question 1 . . . 8
Assignment question 2 . . . 8
Loading node attributes . . . 9
Reading a native R data file . . . 10
The classic: Padgett Florentine Families . . . 10
Assignment task 3 . . . 11
Importing UCINet files directly into R . . . 11
Assignment task 4 . . . 13

Saving Network Data in R **13**
Assignment task 5 . . . 14

3D plots **14**

A network "movie:" *Dragon* by E. Schwartz **14**

Challenge yourself **15**

Recommended literature **16**

References **16**

Acknowledgements **17**

Appendix: Introduction to R and RMarkdown **19**
Installing necessary software . . . 19
Installation order . . . 19

Generating PDFs from within the R Studio . . . 19
Installing R and R Studio . . . 19
R and R studio help guides . . . 19
RMarkdown . . . 20
Learning R . . . 20
Swirl . . . 21
Learning R the old-fashined way . . . 21
The R-Studio . . . 21
Before we begin: The R Help Files . . . 22
Objects in R . . . 23
Operations on Objects . . . 24
Creating objects from operations . . . 25
Working with data . . . 25
Variables and data tables . . . 25
Measurement . . . 27
Nominal and ordinal variables in R . . . 28
Data Frames . . . 29
Other useful information about R . . . 30
Working directory . . . 30
Saving files . . . 30
R Packages . . . 31
Challenge yourself . . . 34

Welcome!

Welcome to the first lecture! This is the supplemental material to the lecture. It contains all the R commands demonstrated in the lecture, and the output they generate. However, it is much more detailed and provides explanations, references and comments - anything and everything that will help you master the material. All literature mentioned in the lecture is included in the "References" section, unless I recommend that you get familiar with it. Then, it is included in the "Recommended literature" section.

For every seminar, there is a series of exercises that you have to perform and turn in for grading. We have discussed the rules in class, so please follow those. To find the exercises, look for Assignment Questions or Assignment Tasks sections. Simply perform the required tasks and answer the questions provided. You can turn them in any readable format, from pdf to Word. If working in groups, make sure you indicate the names of group members at the **top of the file**.

This document also contains a "Challenge yourself" section. It provides additional information and exercises, not discussed in the lecture, should you decide to explore any of the topics further.

As we have discussed in the first lecture, the purpose of this course is to help you *think* differently about data, assuming relational, rather than independent, data points. But this course will also teach you to work with data differently, learning state-of-the-art tools of data analysis and presentation. We will learn network analysis with `R`. Another *"challenge yourself"* aspect is learning additional `R` functionality - creating documents using `RMarkdown`. It is an awesome `R` library, I think, for authoring HTML, PDF, MS Word documents and slide shows. I encourage you to follow my lead and explore this package in addition to learning networks.

Prerequisites

I think by now you have at least some idea of how **'R'** works. If you are just getting started with `R` and still find it challenging, in the Appendix to this file you will find the brief "Introduction to `R`" mini-seminar that might provide tips and tricks you haven't seen before.

No prior knowledge of networks is assumed. Though this first seminar will show some not-so-simple commands, they are shown here just for the reference: we will go over them in much more detail during the course.

Learning objectives

In addition to the learning objectives outlined for the lectures, the goal of this Supplement is to show you how wonderful - and intuitive - the networks are.

Upon completion of the exercises in this Supplement you should be able to do the following:

1. Become familiar the basics of network visualization, including network movie-making.
2. Learn how to load different types of data into 'R' and convert them to network-friendly formats.

Data

All the necessary data files are provided with the lecture. You will need to download them and unzip them. I recommend that you store them all in the same folder. It is also easier that your R file is located in the same folder as the data files.

No matter where you store the files, you will most likely need to set the working directory. So make sure your working directory is set to a folder where the data files are located; otherwise, you will be forced to provide a full path to data every time.

For Lecture 1, you need the following data files:

- "Hartford Drug Users" network, file 'drugnet2.paj'
- "Florentine Families" network in the native R format, file 'flo.Rdata'
- "Dragon" play network, 65 Pajek files in the format 'T1.paj'
- An "attribute" data file for the Dragon play network, "Activity.csv"
- A file for the "Introduction to R" assignment in the Appendix, 'carprise.csv'

Working with 'Materials' files

As I mentioned and you will see, the file includes the `R` code. To simplify your work, I suggest that you create a new `R Script` (in RStudio, it's the sequence File -> New File -> R Script).

I also suggest that you store the new `R Script` file in the same folder where you saved all the data files. This way, you can set the working directory to the "Source file location" as I have demonstrated in the lecture.

To work with code, you can copy the code from this file into your newly created R Script file. Many lines of code are commented with the "#" sign. When the line is "commented out," it provides some information for you, but R ignores it as a valid command. Some of the commented-out items in the code are the true comments, so pay attention when you are removing the "#" signs as to not generate the unnecessary frustration.

I also suggest that you copy the R code into the new file one chunk at a time and follow the explanations provided. This way you can make sure your output matches mine, and if an error is generated, you can quickly find the source.

Required packages

As you most likely know, **'R'** in its basic form can do very little. To implement more advanced analysis, you need to install additional packages. To help you install packages you need for every lecture, we list them at the beginning of each file, so you can get that part out of the way.

For Lecture 1, the following packages are required:

1. ‘sna‘
2. ‘network‘
3. ‘foreign‘
4. ‘igraph‘
5. ‘ndtv‘
6. ‘RColorBrewer‘
7. ‘rgl‘
8. ‘ergm‘

To install a particular package, run the `install.packages("packagename")` command in `R` Console once (notice that the package name should be in quotes. For example, this is what should installation of a package should look like (it is “commented out” for me with a # sign, so that the code does not run in this file):

```
#install.packages("sna") <- # is a comment. Remove it to run the command.
```

Then, all you have to do is execute the `library(packagename)` command when you see it in `R` code. When you call the library, you do not need quotes around the package name (as you will see in my code).

What are networks? Random vs. non-random

We have discussed in the first lecture that much of our work on social network analysis is figuring out whether our network is random or there are some *social* processes behind the network formation. Today, we are going to create a random network to see what it looks like.

Creating a network object from the matrix data

One of the most common ways to represent a network is an *edgelist*, or a list of ties. Once we have imported it into `R`, we can transform it into a network object using the *network* package.

To create a network object, we’ll need a special library. Let’s start by loading the “network” package - the chunk below allows you to install it if you don’t have it on your computer already.

```
# This installs the network package
## install.packages('network')
```

The chunk of code below attaches the first library we will need for our work. To keep this document clean, we added the `suppressPackageStartupMessages` command. Including this command suppresses the regular output that usually accompanies the package loading and is required for diagnostics. Typically, we do not suppress this information because it could be critical for diagnostics; here it’s included for the appearance’s sake, after the package loading was verified to be error-free.

```
# This loads the network package
suppressPackageStartupMessages(library(network))
```

Generating a matrix of data

We could use an existing data matrix, but it could be fun to create our own - so let’s go ahead and do that. We will create a random network - similar to random networks we will later compare our own networks to. We are creating a matrix of data, or sociomatrix - one of the most common types of data.

First, we create a variable that will tell matrix generator how many nodes to store.

```
# Let's have 15 nodes (you can choose any number).
num_nodes <- 15
```

Next, we generate the matrix using the build-in `matrix` command in `R`:

```
my_matrix<-matrix(round(runif(num_nodes*num_nodes)), # edge values
                          nrow = num_nodes, #nrow must be same as ncol
                          ncol = num_nodes)
```

Next, let's make sure there are no self-referencing loops (meaning, the node is not connected to itself):

```
diag(my_matrix) <- 0
# We can check dimensions of the new object:
dim(my_matrix)
```

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

You should see that matrix dimensions are equal to the number of nodes you have specified. You can also check the class of your new dataset:

```
class(my_matrix)
```

```
## [1] "matrix"
```

```
# Let's check whether any data is missing:
sum(is.na(my_matrix))
```

```
## [1] 0
```

There should not be any missing data, because we have generated the matrix ourselves. However, with real-life data, it is not always the case, and with network data especially, missing data could be a problem. We will talk more about this in lectures.

So, now let's go ahead and create an edgelist from our matrix. Command for doing so is `as.network`, and you can learn more about it by reading the help files.

```
# Command below shows the help file for the as.network command.
# Uncomment it to see the info.
##?as.network
my_network<-as.network(x = my_matrix, # the network object
                  directed = TRUE, # specify whether the network is directed
                  loops = FALSE, # do we allow self ties (should not allow them)
                  matrix.type = "adjacency" # the type of input
                  )
```

And now we have the network!

We can do our first tests on our newly created network:

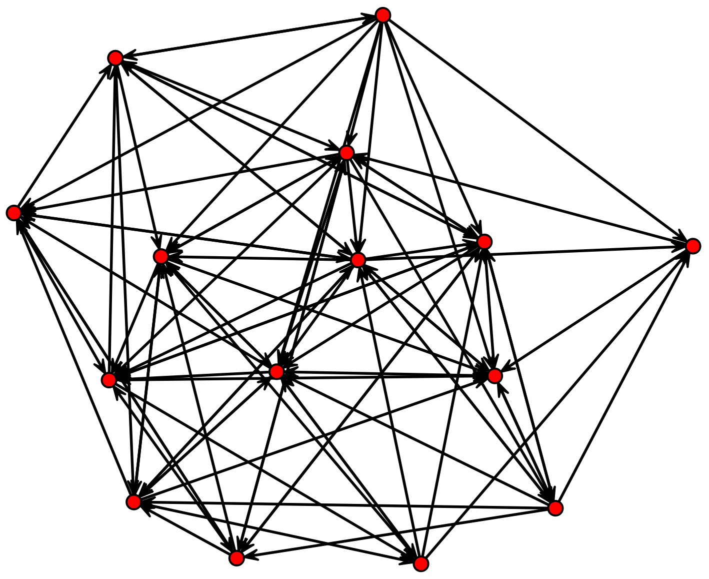
```
network.size(my_network)
```

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

Command below provides all information about the data and shows the matrix, from which we created the data. It could be helpful in case we did not start with the matrix first, but with the edgelist first.

We commented it out, because the output takes a lot of space. Experiment with the command on its own in the console.

```
## summary(my_network)
# View your network:
par(mar=c(1,1,1,1)) # get rid of the large margins and plot the data:
plot(my_network)
```



A picture is worth a thousand words

This section follows the first part of the lecture, “A picture is worth a thousand words.” You are not required to completely understand all the commands that are implemented in this section. Do not worry, they will be thoroughly demonstrated later.

Importing network data in different formats

Network data will generally come as a raw text file or as a file saved in another analysis tool. The package *foreign* allows you to load several data formats with functions that follow the same syntax as to the **read.table()** function in base `R`.

The following sniplet of code installs and provides information about a library “foreign,” which allows you to read other data types into R. Uncomment the code to see what it does.

```
##install.packages("foreign")
##library(foreign)
##?read.dta # reads STATA files
##?read.xport  # reads SAS xport files
```

While *foreign* will help you read data saved in other statistical packages, network data usually come as either raw text or from a specialized network analysis software such as UCINet.

Reading a Pajek file

Pajek is an awesome network program, and if we have time, we’ll look at it, though most of its capabailities are now incorporated in `R`. The package `network` includes the function `read.paj()`, which allows you to load a Pajek project file into `R`.

We will demo `read.paj()`, reading data from a respondent-driven sample (RDS) study of drug users in Hartford, CT. This redacted dataset allows us to observe how the structure network data sampled through RDS differs from the structure of both complete and personal networks. The relation measured is referral into the study.

The classic: Hartford drug users

The first example is a classic example of Hartford Drug users. We start with this example to show how descriptive network analysis can be inferential at the same time.

The Hartford Drug Data

Description. This is a dichotomous adjacency matrix of drug users in Hartford. Ties are directed and represent acquaintanceship. The network is a result of two years of ethnographic observations of people's drug habits.

- Network data. 1-mode matrix 293x293 person by person, directed ties. Relations are acquaintanceship.

- Attribute dataset includes ethnicity and gender.
  Ethnicity codes: 2 = African American; 3 = Puerto Rican/Latino; 1, 5, 6, 7 = white or other
  Gender codes: 1 = male; 2 = female; 0 = unknown.

- Source: https://sites.google.com/site/ucinetsoftware/datasets/covert-networks/drugnet.

The chunk of code below attaches the first library we will need for our work. To keep this document clean, we added the *suppressPackageStartupMessages* command. Including this command suppresses the regular output that usually accompanies the package loading and is required for diagnostics. Typically, we do not suppress this information because it could be critical for diagnostics; here it's included for the appearance's sake, after the package loading was verified to be error-free.

```
suppressPackageStartupMessages(library(sna))
```

The next chunk of code reads the network data and the attributes, and plots the network without the attributes.

```
drugpaj <- read.paj('drugnet2.paj') #read the data
drug <- drugpaj$networks[[1]] # extract network
gender<-drugpaj$partitions[[1]] #extract the attributes
suppressPackageStartupMessages(library(knitr)) #allows for better-looking tables
kable(table(gender), col.names=c("Gender","Frequency"))
```

Gender	Frequency
0	7
1	200
2	86

```
ethnicity <- drugpaj$partitions[[2]]
kable(table(ethnicity), col.names=c("Ethnicity","Frequency"))
```

Ethnicity	Frequency
1	25
2	99
3	155

Ethnicity	Frequency
4	14

Look at the numbers in the tables. What do they tell you? Nothing but frequencies. These are just that - descriptive statistics that, without additional analysis, are hardly informative. We will return to these numbers shortly.

```r
par(mar=c(0,0,0,0))
plot(drug)
```

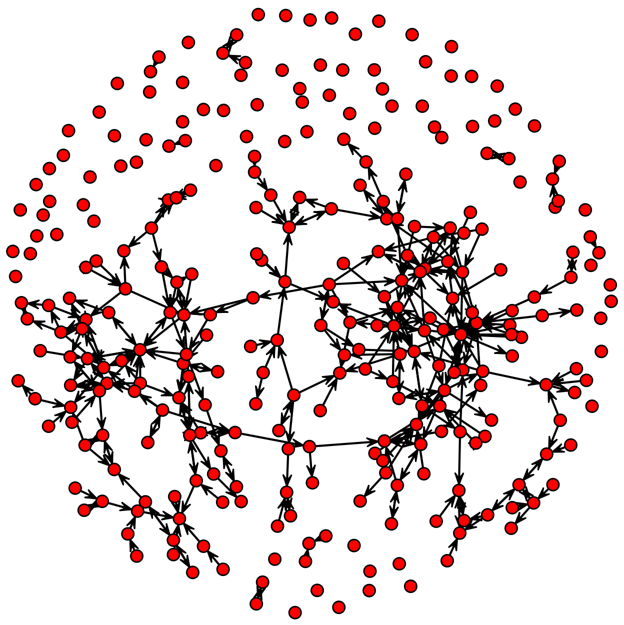


Figure 1: Our first network plot: 'Hartford Drug Users'

Assignment question 1

what looks strange about this network? Why?

You can further examine the network:

```r
network.size(drug) # how many nodes?
```

```
## [1] 293
```

```r
network.edgecount(drug) # how many edges?
```

```
## [1] 337
```

```r
network.dyadcount(drug) # how many dyads?
```

```
## [1] 85556
```

Assignment question 2

What do the numbers above represent?

Loading node attributes

Next, we load the attributes we read above onto the network and generate the graphs with colored attributes.

```
#Set vectors based on attributes.

#Number of node sides allows to create different shapes
#(3=triangle, 4=square, etc.)
sides<-ifelse(ethnicity==1,12, ifelse(ethnicity==2, 3, ifelse(ethnicity==3, 4, 6)))

#Set colors by gender, including gray for unknown:
colors<-ifelse(gender==2,"palevioletred",ifelse(gender==1,"royalblue2","gray8"))

par(mar=c(0,0,0,0)) # And the plot itself:
plot(drug, vertex.col=colors, vertex.sides=sides, vertex.cex=1.5)
```

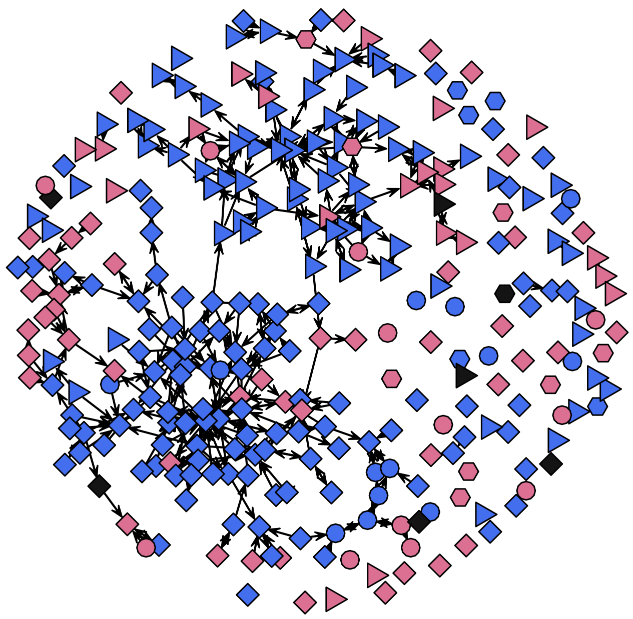


Figure 2: Plot of 'Hartford Drug Users' network with attributes

It was not the best choice of graph presentation, so we did something different:

```
colors2<-ifelse(ethnicity==1,"red", ifelse(ethnicity==2, "green", ifelse(ethnicity==3, "blue", "yellow"
sides2<-ifelse(gender==2,12,ifelse(gender==1,3,4))

par(mar=c(0,0,0,0)) # And the plot itself:
plot(drug, vertex.col=colors2, vertex.sides=sides2, vertex.cex=1)
```

Now, let's talk about frequencies again. Note how much more informative the frequencies are when we are analyzing networks. You can almost immediately make an inference that drug users in this study prefer to be connected to other people of their own races. Same information as in tables, but presented very differently. Graphs, as a tool, belongs to a family of descriptive statistics, but with networks, descriptives quite often become inferential.

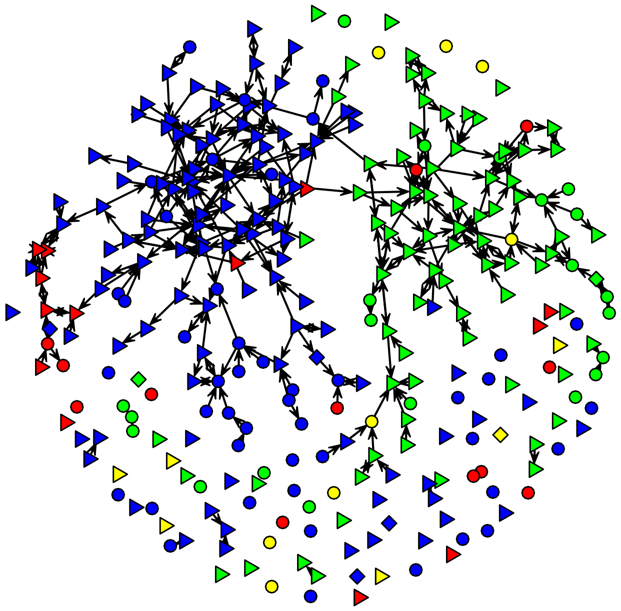


Figure 3: Better visualization of multi-category attributes, 'Hartford Drug Users' network

Reading a native R data file

One of the data packages, the Florentine Family network, we will be working with extensively in this class. Because it comes with R package, it is already in an R format. For the purposes of this class, we have extracted it as a separate file, which is provided in your Seminar 1 folder.

The classic: Padgett Florentine Families

Florentine Families Network

Description. Breiger & Pattison (1986), in their discussion of local role analysis, use a subset of data on the social relations among Renaissance Florentine families (person aggregates) collected by John Padgett from historical documents. Substantively, the data include families who were locked in a struggle for political control of the city of Florence in around 1430. Two factions were dominant in this struggle: one revolved around the infamous Medicis (9), the other around the powerful Strozzis (15).

- Network data. 2 matrices with undirected ties. Relations are marriage and business partnerships.
- Attribute dataset includes:
  1. each family's net wealth in 1427 (in thousands of lira);
  2. the number of priorates (seats on the civic council) held between 1282-1344; and
  3. the total number of business or marriage ties in the total dataset of 116 families.
- Source: https://sites.google.com/site/ucinetsoftware/datasets/padgettflorentinefamilies.

Here are a few pictures of this network from the lecture. We started with loading the data and generating a simple plot:

```r
load('flo.Rdata')
suppressPackageStartupMessages(library(RColorBrewer))
flomarriage <- as.network(as.matrix(flo.marriage), directed=FALSE)
```

```
# Add attributes
set.vertex.attribute(flomarriage, 'wealth', flo.att[,2])
```

Next, we can create simple plots. To more easily distingush the nodes, we can color them different colors. Quite often, we color nodes by attributes. We can also color them randomly.

To do so, we call the *brewer.pal* option from the color package, RColorBrewer (we will examine it in more detail next time). We select node colors from the palettes inside the package. To tell R which palletes we want, we call the palette (‘RdYlBu’). But in front of it, we tell R which nodes to color from that palette.

We have a total of 20 nodes, so we color first 11 (brewer.pal(11,‘RdYlBu’)) with one set of colors, and the second 9 with another. Also notice that we store the colors in the vector, so that we can use it later.

```
FloColors <- c(brewer.pal(11,'RdYlBu'),brewer.pal(9,'RdYlBu')) #set a vector of colors
```

Now we are ready to draw the network:

```
par(mar=c(0,0,0,0))
plot(flomarriage,
     vertex.cex=(get.vertex.attribute(flomarriage, 'wealth')/25 +.4),
     displaylabels=TRUE,
     label.cex=.5,
     label.pos=0, vertex.col=FloColors)
```

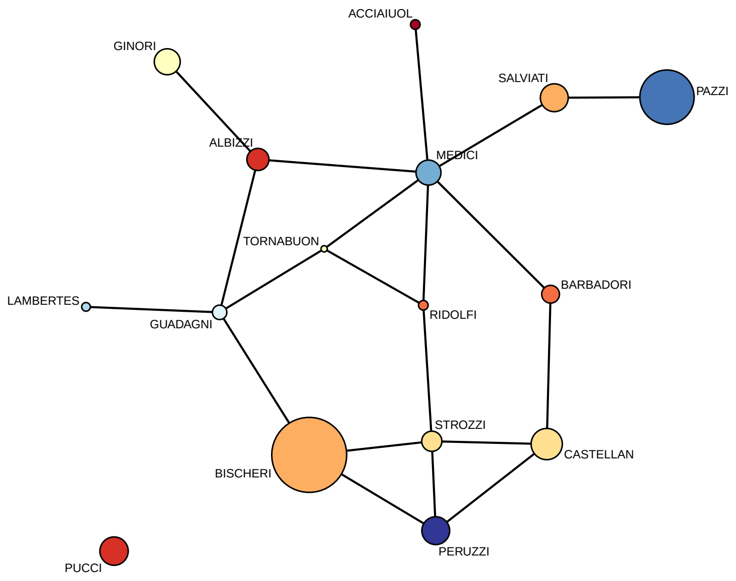


Figure 4: Florentine Families network with ’wealth’ as the vertex size

Assignment task 3

You may have noticed that there is another network in that data file - the business network. Please plot the biz network with node attributes that you’ve set above.

Importing UCINet files directly into R

UCINet files are space-delimited text files with a header. If you inspect the file, you can figure-out how to read it in R:

1. Open the UCINet file in 'R studio' (as a text file)
2. Note the line number where the data actually starts
3. Read the file into R starting at that line number
4. Use information from the header to name variables

We will demo this process with Padgett's Florentine Families data, which we've seen already, and which is included as a part of a different set of data. Reading commands are different for .dat types of files, so please carefully step through the code below:

```
# Read vertex labels with scan()
flo.names <- scan('padgett.dat', what='character', skip=4, nlines=16)
# Read data with read.table()
flos <- read.table('padgett.dat', skip=41, col.names=flo.names)
# Read node attributes:
flo.att <- read.table('padgw.dat',
col.names =c('WEALTH','NUM.PRIORS','NUM.TIES'), skip=25)
flo.att
```

```
##    WEALTH NUM.PRIORS NUM.TIES
## 1      10         53        2
## 2      36         65        3
## 3      27         38        4
## 4     146         74       29
## 5      55          0       14
## 6      44         12        9
## 7      20         22       18
## 8       8         21       14
## 9      42          0       14
## 10    103         53       54
## 11     48          0        7
## 12     49         42       32
## 13     10         35        5
## 14     48          0        7
## 15     32          0        9
## 16      3          0        1
```

Please take a look at the flo.att data. As you can tell, it is simply a set of data, which are all attributes. How do we add it to network? Check the code below.

```
flo.att <-cbind(flo.names,flo.att)
```

Command below provides first six rows of data, so that you can check your data without loading the entire dataset.

```
head(flo.att)
```

```
##   flo.names WEALTH NUM.PRIORS NUM.TIES
## 1 ACCIAIUOL     10         53        2
## 2   ALBIZZI     36         65        3
## 3 BARBADORI     27         38        4
## 4  BISCHERI    146         74       29
## 5 CASTELLAN     55          0       14
## 6    GINORI     44         12        9
```

In this particula network setup, marriage network and business networks are joined in one file. It could happen and with other data you use; sometimes, you need to separate the files. Code below shows how to break up files into needed chunks.

```
# Separate adjacency matrices
# subset of the first 16 colums is the marriage network
flo.marriage <-flos[1:16,]
dim(flo.marriage)
```

```
## [1] 16 16
```

```
row.names(flo.marriage) <-flo.names # name
flo.biz <- flos[17:32,] # subset of the second 16 is the business network.
row.names(flo.biz) <-flo.names # name
dim(flo.biz)
```

```
## [1] 16 16
```

```
# Check the data by listing a couple of rows and colums from each network.
flo.marriage[1:2,1:2]
```

```
##            ACCIAIUOL ALBIZZI
## ACCIAIUOL          0       0
## ALBIZZI            0       0
```

```
flo.marriage[15:16,15:16]
```

```
##            STROZZI TORNABUON
## STROZZI          0         0
## TORNABUON        0         0
```

```
flo.biz[1:2,1:2]
```

```
##            ACCIAIUOL ALBIZZI
## ACCIAIUOL          0       0
## ALBIZZI            0       0
```

```
flo.biz[15:16,15:16]
```

```
##            STROZZI TORNABUON
## STROZZI          0         0
## TORNABUON        0         0
```

Now that matrices are separated and set up in the way we need them to be set up, let's create networks from data and add attributes to nodes.

```
flo.marriage <- as.network(as.matrix(flo.marriage),directed=FALSE)
flo.biz <- as.network(as.matrix(flo.biz),directed=FALSE)
## add attributes
set.vertex.attribute(flo.marriage, 'wealth', flo.att[,2])
set.vertex.attribute(flo.biz,'wealth', flo.att[,2])
```

Assignment task 4

Using the code already shown, plot both of the new networks. Add attributes if you wish.

Saving Network Data in R

If you will be using the dataset in R, it's best to save as `filename.Rdata` but if you plan on sharing the data, then you will want to save it as text.

```
# Save several objects in the same .Rdata file to load all at once
save(flo.marriage, flo.biz, file='floNets.Rdata')
# Save network as an edgelist in a .csv file
drug.edges <- as.matrix(drug, matrix.type='edgelist')
write.csv(drug.edges, file='drugsEdgelist.csv',row.names=FALSE)
```

Assignment task 5

For the network "drug" that we created and loaded with attributes, create several different network plots, adding gender and ethnicity to the graph as node attributes. Consider using a variety of colors to make your plot more informative.

3D plots

Next, in the lecture we drew several 3D plots. Because they open in a different window (to allow moving the network manually), we can't include them in the file. Therefore, here is the code, but the actual plots are commented out (with a # sign). Remove the *single* # sign when copying the code into your R console (double ## sign is reserved for comments), and it will work.

```
##A simple 3D plot:

#gplot3d(flomarriage, vertex.col=FloColors,
#        vertex.radius=1.5, edge.lwd=.2)

##Add names

#gplot3d(flomarriage, vertex.col=FloColors,
#        vertex.radius=1.5, edge.lwd=.2, displaylabels=TRUE)

##Add wealth

#gplot3d(flomarriage, vertex.col=FloColors,
#vertex.radius=(get.vertex.attribute(flomarriage, 'wealth')/30 +.5),
#edge.lwd=.2, displaylabels=TRUE)
```

A network "movie:" *Dragon* by E. Schwartz

Evgeny Schwartz wrote the play "The Dragon" in 1944. A reference to the English translation is provided in the References section, if you want to familiarize yourself with it.

It is seen as a satire on the political climate of USSR at that time. The play is a story of the knight Lancelot, who came to liberate the city from the Dragon - an evil creature torturing the city's citizens and killing one beautiful woman every year. Lancelot accomplishes his mission, but he finds that Dragon was a just cover for the bureaucratic hierarchy of the city's corrupt government clenching to power.

Network of "Dragon" - play by E. Schwartz

Description. The network for this movie was created by HSE University Lyceum student Daniel Kuskov. He coded the play into scenes based on character changes. Every time a character left a scene of the play or a new character appeared, a new network - with active actors - was created. Because each scene is a natural progression through the play, it imitates the time-series networks - a change of networks over time.

- Network data. 65 Pajek data files, each corresponding to a play in the scene.
- Attribute data are counts of the times that each character appeared in the play prior to the current scene.
- Source: https://anr.hse.ru.

The code below shows all the steps necessary to create the movie. However, because the RMarkdown document is static, the movie command itself is commented out.

```
suppressPackageStartupMessages(library(ndtv))
suppressPackageStartupMessages(library(foreign))
#Create a list object to contain all individual network time-slices
drakon=list()
for (i in 1:65) {
  drakon[[i]]<-read.paj(paste("T",i,".paj",sep=""))
}
drakNet<-networkDynamic(network.list=drakon, start=1)
```

```
## Onsets and termini not specified, assuming each network in network.list should have a discrete spell
## Argument base.net not specified, using first element of network.list instead
## Created net.obs.period to describe network
##  Network observation period info:
##   Number of observation spells: 1
##   Maximal time range observed: 1 until 66
##   Temporal mode: discrete
##   Time unit: step
##   Suggested time increment: 1
```

```
activity<-read.csv("Activity.csv")
activity<-activity[,4:68]
for (i in 1:65) {
  activate.vertex.attribute(drakNet,'activity',activity[[i]],onset=i,terminus=i)}

## Uncomment the code below to run the movie
#render.d3movie(drakNet, plot.par=list(displaylabels=T,
#vertex.col=FloColors, vertex.cex = function(slice){slice%v%'activity'/5+0.5}))
```

Challenge yourself

Here is the first assignment that is not graded, but could help aid your understanding of the material. Have fun!

1. We talked in class that for network analysis, the lines between descriptive and inferential analysis is blurred: most descriptive measures are automatically inferential. Examine the drug network we've drawn in this seminar. Does anything look odd to you?
2. For the drug network, draw the network with nodes colored by race. What do you conclude?
3. Take any of the slices from the Dragon set of files and create a network. Add node names, color nodes

randomly to make them differ from each other. Plot the network.

Recommended literature

- Kadushin, C., 2012. Understanding social networks: Theories, concepts, and findings. Oup Usa.
- Robins, G., 2015. Doing social network research: Network-based research design for social scientists. Sage.
- Wasserman, S. and Faust, K., 1994. Social network analysis: Methods and applications.

References

1. Adams, J.S., 1963. Towards an understanding of inequity. The Journal of Abnormal and Social Psychology, 67(5), p.422.
2. Breiger R. and Pattison P. (1986). Cumulated social roles: The duality of persons and their algebras. Social Networks, 8, 215-256.
3. Borgatti, S.P. and Everett, M.G., 1992. Notions of position in social network analysis. Sociological methodology, pp.1-35.
4. Burt, R.S., 1987. Social contagion and innovation: Cohesion versus structural equivalence. American journal of Sociology, 92(6), pp.1287-1335.
5. Burt, R.S., 1992. Structural holes. Harvard university press.
6. Burt, R.S., 2005. Brokerage and closure: An introduction to social capital. Oxford university press.
7. Burt, R.S., Jannotta, J.E. and Mahoney, J.T., 1998. Personality correlates of structural holes. Social Networks, 20(1), pp.63-87.
8. Buskens, V. and Van de Rijt, A., 2008. Dynamics of networks if everyone strives for structural holes. American Journal of Sociology, 114(2), pp.371-407.
9. Freeman, L.C., 1992. The sociological concept of" group": An empirical test of two models. American journal of sociology, 98(1), pp.152-166.
10. Greenberg, J.R. and Greenberg, P.J., 1991. Oedipus and beyond: A clinical theory. Harvard University Press.
11. Haidt, J. and Rodin, J., 1999. Control and efficacy as interdisciplinary bridges. Review of general psychology, 3(4), pp.317-337.
12. Heider, F., 1946. Attitudes and cognitive organization. The Journal of psychology, 21(1), pp.107-112.
13. Homans, George C. "The Human Group. New Brunswick." (1950).
14. Kadushin, C., 2002. The motivational foundation of social networks. Social networks, 24(1), pp.77-91.
15. Kadushin, C. and Jones, D.J., 1992. Social networks and urban neighborhoods in New York City. City & Society, 6(1), pp.58-75.
16. Kalish, Y. and Robins, G., 2006. Psychological predispositions and network structure: The relationship between individual predispositions, structural holes and network closure. Social networks, 28(1), pp.56-84.
17. Kent D. (1978). The rise of the Medici: Faction in Florence, 1426-1434. Oxford: Oxford University Press.
18. Lin, N., 1990. Social resources and social mobility: A structural theory of status attainment. Social mobility and social structure, 3, pp.247-261.

19. Lin, N., 2008. A network theory of social capital. The handbook of social capital, 50(1), p.69.

20. Martin, A.M. and Carey, E., 2009. Person-centred plans: empowering or controlling?. Learning Disability Practice (through 2013), 12(1), p.32.

21. Moody, J., 2001. Race, school integration, and friendship segregation in America. American journal of Sociology, 107(3), pp.679-716.

22. Newman, A.M. and Cooper, J.B., 2010. AutoSOME: a clustering method for identifying gene expression modules without prior knowledge of cluster number. BMC bioinformatics, 11(1), pp.1-15.

23. Podolny, J.M. and Baron, J.N., 1997. Resources and relationships: Social networks and mobility in the workplace. American sociological review, pp.673-693.

24. Robins, G., Pattison, P. and Elliott, P., 2001. Network models for social influence processes. Psychometrika, 66(2), pp.161-189.

25. Sandefur, R.L. and Laumann, E.O., 1998. A paradigm for social capital. Rationality and society, 10(4), pp.481-501.

26. Schwartz, E.L. Dragon (1944). Translated by Y. Machkasov. http://a7sharp9.com/dragon.htm.

27. Weeks, Margaret R., Scott Clair, Stephen P. Borgatti, Kim Radda, and Jean J. Schensul. "Social networks of drug users in high-risk sites: Finding the connections." AIDS and Behavior 6, no. 2 (2002): 193-206.

28. Internet resources for images, information, and other references in the lecture:
    - The map of Moscow rivers: http://www.analytictech.com/mb119/pitts.htm
    - The Roman road system network: http://www.fh-augsburg.de/~harsch/Chronologia/Lspost03/Tabula/tab_pe05.html
    - The protein molecule: http://www.rcsb.org/
    - The real neural network of a roundworm: https://archive.nytimes.com/www.nytimes.com/interactive/2011/06/20/science/brain.html
    - The Wine network: https://tinyurl.com/TheWineNetwork
    - Internet as a network: http://cheswick.com/ches/map/gallery/index.html;http://cheswick.com/ches/map/gallery/index.html
    - The NY Times archive of Dr. J.L.Moreno study https://www.nytimes.com/1933/04/03/archives/emotions-mapped-by-new-geography-charts-seek-to-portray-the.html
    - The "Money Network" https://dealbook.nytimes.com/2011/04/07/the-money-network/

Acknowledgements

In preparing this lecture, I have relied on knowledge and materials accumulated over the course of my long career. I have learned networks from many different courses. In preparation of materials for this course, I have relied on what I have learned, and on some of the information available freely on the internet and elsewhere.

My special gratitude goes to:

- Professor Stanley Wasserman, the founder of the International Laboratory for Applied Network Research (home to this course), who taught me the networks. I have referred to his materials, and continue to do so, on a regular basis. https://psych.indiana.edu/directory/faculty/wasserman-stanley.html

- Professor Vladimir Batagelj, current scientific supervisor of the Laboratory and a faculty member of this program. I have used some of his work (with acknowledgment) in this lecture. https://www.hse.ru/en/org/persons/222862219
- Professor Katherine Ognyanova, who I do not know, but whose network visualization vignettes (freely available on the internet) were very useful in creating some of my own assignments. https://kateto.net/network-visualization.

Appendix: Introduction to R and RMarkdown

There are many ways to start learning R. We suggest that you use Swirl as an interactive option to learn R right in the R-studio environment. Alternatively, we have provided brief instructions in this document (to also demontrate capabilities of RMarkdown). No matter the way you go, R is a fun and very powerful tool, and we hope you become familiar with it enough to use it long after this class ends, for all of your computing needs.

Important: Installation may not be easy. There could be some issues with version compatibility, time delays with slow internet, etc. Please carefully follow the instructions below to avoid the many problems that may come up.

Installing necessary software

R is a very powerful statistical instrument that (through open source and user support) has by far exceeded capabilities of other programs such as SPSS, SAS and Stata. It used to require a lot of programming, but now there are many user-written packages that do the nitty gritty work for us. There is still a bit of a learning curve, but we will take it one step at a time, starting with the basics.

Installation order

If you do not have any of the required software on your machines, then install in the following order:

1. R first
2. LaTex files second
3. R-studio last

We do not have an order preference, and for that reason, installation of LaTex files is covered first. Again, this is optional for those who want to use RMarkdown to create documents. What is mandatory, however, is that you install R-studio AFTER you install R.

Generating PDFs from within the R Studio

There are several ways to generate PDFs in R Studio, but since RMarkdown has that functionality already, you will only need to install the necessary libraries for LaTex language support. We use MikTex, but apparently, the file is very heavy and could take quite some time to download.

Installing R and R Studio

Go to http://swirlstats.com/students.html. Install (in that order!) R, then R-studio. Make sure you install a free version of R Studio (not server; we won't need it for this class).

R and R studio help guides

There are many, many references that will help you learn R. The list below is just to get you started; you should be able to find ample sources on Google if you need more help. For now, these are just basics; we'll start working with network packages next week.

1. https://cs.hse.ru/data/2015/04/03/1096436989/Torfs+Brauer-Short-R-Intro.pdf
2. https://cs.hse.ru/data/2015/05/13/1098775155/R%20Tutorial.pdf
3. https://cs.hse.ru/data/2015/04/03/1096437593/R-refcard.pdf
4. https://support.rstudio.com/hc/en-us
5. http://www.statmethods.net/
6. http://www.burns-stat.com/pages/Tutor/hints_R_begin.html
7. http://data.princeton.edu/R/gettingStarted.html
8. http://www.ats.ucla.edu/stat/R/sk/
9. Of course, just as for any data analytics nowdays, there is a Coursera course for R as well: https://www.coursera.org/learn/r-programming

RMarkdown

RMarkdown is simply awesome. Once you learn to use it, you will never go back to other editing software, we promise! Despite its simplicity, it's quite powerful and allows you to take advantage of best features of R (you can embed fully functioning code chunks) and LaTex (you can create beautiful documents); you can create HTML, PDF, and Word outputs with the same script. No other software, as far as we know, has this capacity.

To install R Markdown, go here: http://rmarkdown.rstudio.com/. The site has full instructions, though just as Swirl, Markdown is installed as a package - just follow the instructions. The very first Markdown template that gets generated upon installation will have a set of basic instructions, and the button you will learn to love, the **Knit** button, will generate a document with both the content and the embedded R chunks.

There are many useful hints right on that website. In addition, you will find other useful references and links, such as the ones below, to help you learn to format and render your documents just the way you like them. We will greatly appreciate the time you take to learn new tips and tricks, and will give extra-credit points to assignments that go above and beyond the very basic requirements that we specify for each assignment.

Here are some useful references for R Markdown:

1. http://www.stat.cmu.edu/~cshalizi/rmarkdown/#mark-up-markdown
2. https://cs.hse.ru/data/2015/04/03/1096437543/1501.01613v1.pdf
3. https://www.rstudio.com/wp-content/uploads/2016/03/rmarkdown-cheatsheet-2.0.pdf
4. https://www.rstudio.com/wp-content/uploads/2015/03/rmarkdown-reference.pdf
5. http://web.stanford.edu/class/psych252/section/Rmarkdown_info.html
6. RMarkdownhelpinRussian:http://dkhramov.dp.ua/Comp/FromMarkdownToLaTeX

Have fun exploring this wonderful new tool!

Learning R

There are many, many ways to get started. We recommend that you use Swirl, a package that will work right from inside the R. There is also some literature in Russian to help you, and to demonstrate capabilities of R Markdown, we also included several basic commands with explanations at the end of this file. No matter the way, do not wait until the last possible moment to start working on learning R! Otherwise, you will quickly fall behind.

Swirl

Swirl is just a package, similar to all other packages we will be running, and it is step 3 on the web site above. During the first seminar, we will show you how to install package in R and run it, though there is nothing more to it than simply to run the code at the prompt, as shown on the web site:

```
#install.packages("swirl")
# Code is commented out to not confuse the Markdown )))
```

Then, after the package is installed, type (after the ">" prompt, which shows up automatically)

```
#library("swirl")

##hit Enter, then

#swirl()
```

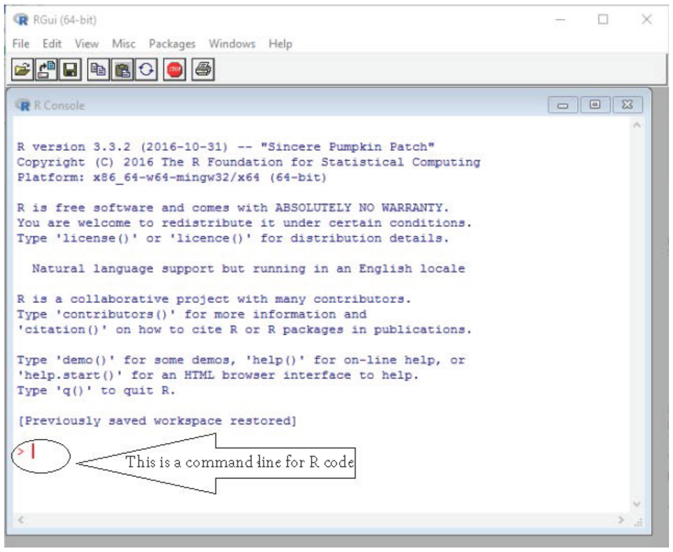
and hit Enter again to run Swirl. To start, complete basic Swirl lessons in R programming: 1, 3-8 (sufficient for now), the rest - optional.

Learning R the old-fashined way

The "old-fashioned" means with instructions in this file. It's just a starter, but should be enough to master this course.

The R-Studio

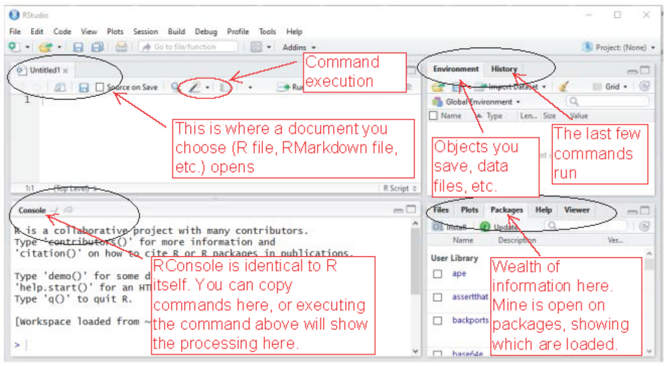
Before R-studio, the user interface for R, was created, all we had was the RConsole - simply a window for writing the code. Some of us still use RConsole when we do heavy-duty programming, because RStudio code sometimes interferes with some user-created functions. But the RConsole simply looked like this:


RGui (64-bit)
File Edit View Misc Packages Windows Help
R Console
This is a command line for R code

```
R version 3.3.2 (2016-10-31) -- "Sincere Pumpkin Patch"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Previously saved workspace restored]

>
```

The RConsole is where you type all your commands, and simply hitting "Enter" will execute the command. Please note that every new line in R starts with the ">" (greater than) sign. It's a signal for R to start a new line, and a signal for you that the code has been executed correctly. If you do not see the ">" sign after an executed command, but see a plus sign ("+") instead, it means that the code line above was incomplete.

The RConsole alone was not very user-friendly; a better version of it, called RStudio, was released several years ago.


Command execution
This is where a document you choose (R file, RMarkdown file, etc.) opens
Objects you save, data files, etc.
The last few commands run
RConsole is identical to R itself. You can copy commands here, or executing the command above will show the processing here.
Wealth of information here. Mine is open on packages, showing which are loaded.

```
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some d
'help.start()' for an HT
Type 'q()' to quit R.

[Workspace loaded from ~

>
```

The R-Studio has a lot of features, which make our lives much easier. Some of the most common we will discuss in class, but please feel free to explore on your own.

Before we begin: The R Help Files

R has built-in help, which simply starts with a question mark. In the help-page for each function (called by ?functionName) you can find how to use a function including the required input types and the default specifications. The good thing about r is that it is very "google-able". If you want to calculate the e.g. standard deviation, just search in google: "calculate standard deviation in R." You can also do a keyword-search the following way, but we had better experience with google.

The code below is commented out to keep RMarkdown document clean. Remove the ## from the code and copy it into your RConsole in order to see the results that the execution produces. Note where in RStudio the answers appear.

For exampme, the mean() command calculates the mean of an object (often a vector/ series of numbers. check also the standard deviation: note that we did not know what the command was, so we simply typed the name of the function we needed.

```
## ?mean
## ??standarddeviation

# If you know or remember the command, but not how to use it, type:

## ?sd
```

For a list and details about available functions see help(Arithmetic), help(Trig), help(complex), help(log), help(Round), help(sqrt), etc.

The R Basics

Below are some instructions and commands in R to help you get started right from this file.

```
print("Hello World")
```

```
## [1] "Hello World"
```

"Print" is a command, or a function. Functions need one or more arguments that you write in parentheses behind the functionName(). Note that R is case-sensitive, i.e. Print("Hello World") does not work (capital P in the command "Print").

R can work as a calculator. Try the following commands:

```
1+5
```

```
## [1] 6
```

```
105*99+6
```

```
## [1] 10401
```

```
exp(0.09855)
```

```
## [1] 1.10357
```

```
mean(c(1, 5, 8, 7, 6, 4, 22, 1, 0.9))
```

```
## [1] 6.1
```

In the normal case, however, you will want to use a script in order to retrace what you have done, be able to repeat what you have done, and to be able to share what you have done.

Objects in R

We generally work with so called objects in R. An object can be a lot different things. You can think of an object as a container, in which you can put whatever you want (numbers, matrices, words, functions…). Objects have a name and content. To store a computed value we use an assignment statement. The "<-" assigns a value to an object. Here are some very basic objects with values assigned to them:

```
NumberA <- 5

NumberB <- exp(2)

VectorC <- c(1,5,8,7,6,4,22,1,0.9,500)
```

c(1,5,8,7,6,4,22,1,0.9) is a vector that is used for these commands. Vectors with a series of integers can be obtained by the colon:

```
VectorD <- 5:15
```

or by the sequence command

```
quarters <- seq(0, 5, by=0.25)
```

we can also create matrices:

```
MatrixE <- matrix(0, 5, 5)
MatrixF <- matrix(1:4, 6, 6)
```

Each of these objects can be called by their name:

```
NumberA
```

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

```
NumberB
```

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

```
MatrixF
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    3    1    3    1    3
## [2,]    2    4    2    4    2    4
## [3,]    3    1    3    1    3    1
## [4,]    4    2    4    2    4    2
## [5,]    1    3    1    3    1    3
## [6,]    2    4    2    4    2    4
```

Objects are the core of R. When using R to analyze networks, everything we do are operations on objects. Our data is stored in an object, the algorithm to estimate, the effects we include…

Operations on Objects

We can do the same type of operations we do we do in the console also with these objects:

```
NumberA*NumberB
```

```
## [1] 36.94528
```

The content of one object is multiplied with the content of another object.

```
NumberA*VectorC
```

```
##  [1]    5.0   25.0   40.0   35.0   30.0   20.0  110.0    5.0    4.5 2500.0
```

```
mean(VectorC)
```

```
## [1] 55.49
```

Most of the time, operations can have multiple arguments if the operation is executed on multiple objects or where you can specfify some more parameters. Multiple arguments are separated by a comma ","

```
intersect(VectorC, VectorD)
```

```
## [1] 5 8 7 6
```

```
mean(VectorC)
```

```
## [1] 55.49
```

```
mean(VectorC, trim = 0.1)
```

```
## [1] 6.75
```

We can also nest operations within one another:

```
mean(intersect(VectorC,VectorD))
```

```
## [1] 6.5
```

Creating objects from operations

We can create new objects from these operations, which we do extensively, because we don't want to create all objects "on foot":

```
product <- NumberA*VectorC
product
```

```
##  [1]    5.0   25.0   40.0   35.0   30.0   20.0  110.0    5.0    4.5 2500.0
```

```
otherNumber <- NumberA*var(VectorC)
otherNumber
```

```
## [1] 122158.6
```

Working with data

The chunk below was written by professor Vladimir Batagelj (https://www.hse.ru/en/org/persons/222862219).

The basic data types in R are: numeric, integer, complex, logical, character, and raw. They can be combined into structured objects using: vector, matrix, array, list, etc.

For testing and converting types we have functions:

1. is.type(x) - is the object x of type type ?
2. as.type(x) - try to convert object x into object of type type.
3. typeof(x) - returns the type of x; similar class(x) and mode(x).

R is a programming language. About the basic control statements see Prof. Batagelj's existing presentation http://tinyurl.com/Vlado-R-Controls.

Variables and data tables

In a standard setting we have an (ordered) set of **units**. To describe them we select a set of their **properties** (attributes). Data are obtained by **measuring** the properties of units. Here, we have a data table that contains information on **units** (in rows) and their **attributes** (in columns).

	P_1	P_2	P_3	$\cdots$	P_m
U_1	$v_{1,1}$	$v_{1,2}$	$v_{1,3}$	$\cdots$	$v_{1,m}$
U_2	$v_{2,1}$	$v_{2,2}$	$v_{2,3}$	$\cdots$	$v_{2,m}$
U_3	$v_{3,1}$	$v_{3,2}$	$v_{3,3}$	$\cdots$	$v_{3,m}$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
U_n	$v_{n,1}$	$v_{n,2}$	$v_{n,3}$	$\cdots$	$v_{n,m}$

In statistics, vectors - columns in the data table are called **variables**. Because of this in R the basic data structure is a vector. Most operations and functions are working vector-wise. Try the following code:

```
a <- c(1,2,3,4,5)
a
```

```
## [1] 1 2 3 4 5
```

```
b <- c(10,9,8,7,6)
b
```

```
## [1] 10  9  8  7  6
```

```
# To find the fourth element in b:
b[4]
```

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

```
b[c(1,3,5)]
```

```
## [1] 10  8  6
```

```
length(a) # Gives you the number of elements in a
```

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

```
# Some more simple operations
a+b
```

```
## [1] 11 11 11 11 11
```

```
a-b
```

```
## [1] -9 -7 -5 -3 -1
```

```
a*b
```

```
## [1] 10 18 24 28 30
```

```
sqrt(a)
```

```
## [1] 1.000000 1.414214 1.732051 2.000000 2.236068
```

```
2**a # Notice multiplying number by a vector
```

```
## [1]  2  4  8 16 32
```

```
# Now, look at several sum/summary commands:
summary(b)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       6       7       8       8       9      10
```

```
sum(b)
```

```
## [1] 40
```

```
cumsum(b)
```

```
## [1] 10 19 27 34 40
```

```
#And some more manipulations:
integer(10)
```

```
##  [1] 0 0 0 0 0 0 0 0 0 0
```

```
1:9
```

```
## [1] 1 2 3 4 5 6 7 8 9
```

```
rep(c(0,1),5)
```

```
##  [1] 0 1 0 1 0 1 0 1 0 1
```

```
seq(1,3,1/3)
```

```
## [1] 1.000000 1.333333 1.666667 2.000000 2.333333 2.666667 3.000000
```

```
c(rep(0,5),rep(1,7))
```

```
##  [1] 0 0 0 0 0 1 1 1 1 1 1 1
```

Measurement

Not all properties of units can be measured, as in physics and geometry, by real numbers. The measurements are made in different **measurement scales** (You can find more information here: https://www.cambridge.org/core/books/measurement-theory/7D75B72C3E5FA676EA7AD6AB4D8DF4A7).

- **Numerical** scales
  - **absolute** (counts)
  - **interval** (temperature F and C, date)
  - **ratio**, or having absolute zero (weight, length, price, temperature K))
- **Ordinal** (grades: excellent, good, average, poor, unsatisfactory; temperature: very cold, cold, cool, warm, hot, very hot)
- **Nominal or Categorical** (nationality, religious preference: Buddhist, Muslim, Christian, Jewish, Other)

The type of the measurement scale determines what we can do with the corresponding variables. For example - central element: geometric mean, (arithmetic) mean, median, mode.

Nominal and ordinal variables in R

Values of numerical scale measurements are represented with integers or real numbers. Integers are often used (as codes) also for representing ordinal and nominal values - but not all numerical operations on them are meaningful.

```
# Notice what the following code is doing:
t <- c("F","D","F","I","A","F","F","D","B","SLO","I","F","GB","B")
T <- factor(t)
t
```

```
##  [1] "F"   "D"   "F"   "I"   "A"   "F"   "F"   "D"   "B"   "SLO" "I"   "F"
## [13] "GB"  "B"
```

```
T
```

```
##  [1] F   D   F   I   A   F   F   D   B   SLO I   F   GB  B
## Levels: A B D F GB I SLO
```

```
as.integer(T)
```

```
##  [1] 4 3 4 6 1 4 4 3 2 7 6 4 5 2
```

```
levels(T)
```

```
## [1] "A"   "B"   "D"   "F"   "GB"  "I"   "SLO"
```

```
levels(T)[as.integer(T)]
```

```
##  [1] "F"   "D"   "F"   "I"   "A"   "F"   "F"   "D"   "B"   "SLO" "I"   "F"
## [13] "GB"  "B"
```

```
which(T=="F")
```

```
## [1]  1  3  6  7 12
```

```
table(T)
```

```
## T
##   A   B   D   F  GB   I SLO
##   1   2   2   5   1   2   1
```

```
L <- c("unsatisfactory","poor","average","good","excellent")
s <- c("unsatisfactory","good","good","average")
k <- factor(s,levels=L,ordered=TRUE)
k
```

```
## [1] unsatisfactory good           good           average
## Levels: unsatisfactory < poor < average < good < excellent
```

```
as.integer(k)
```

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

R is also unicode. That means, you can code your variables in Russian if you choose to do so.

Data Frames

Data tables are represented in R as data frames (R code is “data.frame”). Data frame is essentially a special list in which all items are vectors (columns, variables) are of the same length. A data frame can be created with an assignment of the form:

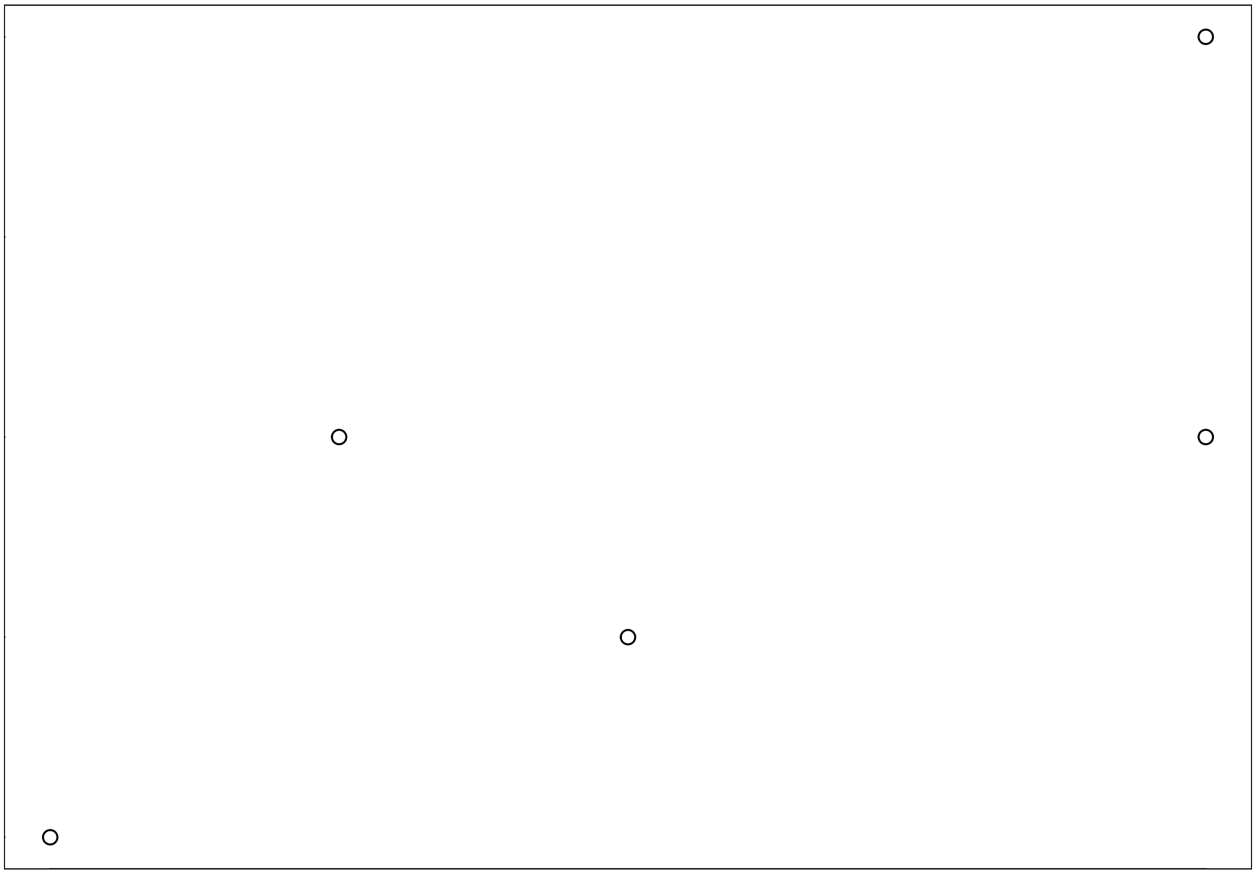
D <- data.frame(e1, e2, e3, . . . , ek, row.names=units)

where “units” is a vector containing names of units, and e_i has either a form $n_i = v_i$ or v_i (where n_i is the name of the $_i$-th column and v_i is the vector of values).

```
units <- c('a','b','c','d','e')
u <- c(1,2,3,5,5)
v <- c(1,3,2,3,5)
D <- data.frame(U=u,V=v,row.names=units)
D
```

```
##   U V
## a 1 1
## b 2 3
## c 3 2
## d 5 3
## e 5 5
```

```
par(mar=c(0,0,0,0))
plot(D) # this will give you a picture
```



We can get the $_i$th variable by expression D[[i]].

```
# Extract the second item in the data frame:
D[[2]]
```

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

A data frame D can be created also interactively using

D <- edit(data.frame())

We can also prepare it as an Excel spreadsheet, save it as a CSV file, and read it into R (we'll talk about it in one of the following labs).

Other useful information about R

Working directory

R needs to know where to find files on your computer. Usually there is a default working directory that you can find using the getwd() command. We can change it using the function setwd(*[path]*) command or going to Session->Set Working Directory menu option in RStudio.

Saving files

In addition to familiar *Save* and *Save as* commands in the File menu, you can also use the *save*.Rdata using the command dump(c(“v1”,“v2”,. . . ,“vk”),“save.Rdata”)

When we need these files again, we can call them using

source(“save.Rdata”)

R Packages

The Basic R has only the limited number of functions. There are many, many more that have been added by users. Because not all the code is required by everyone, expanding the software itself is unreasonable. Therefore, people created a variety of packages that contain only the functions that they would need for a certain task. We will be using several main SNA packages, and at the beginning of every lab, we will list the necessary packages (which you add using the ‘install.packages’ command) for that particular lab.

R packages also contain data. Explore a few, as is shown in the code below:

```
data(package=.packages(all.available=TRUE))
```

```
## Warning in data(package = .packages(all.available = TRUE)): datasets have been
## moved from package 'base' to package 'datasets'

## Warning in data(package = .packages(all.available = TRUE)): datasets have been
## moved from package 'stats' to package 'datasets'
```

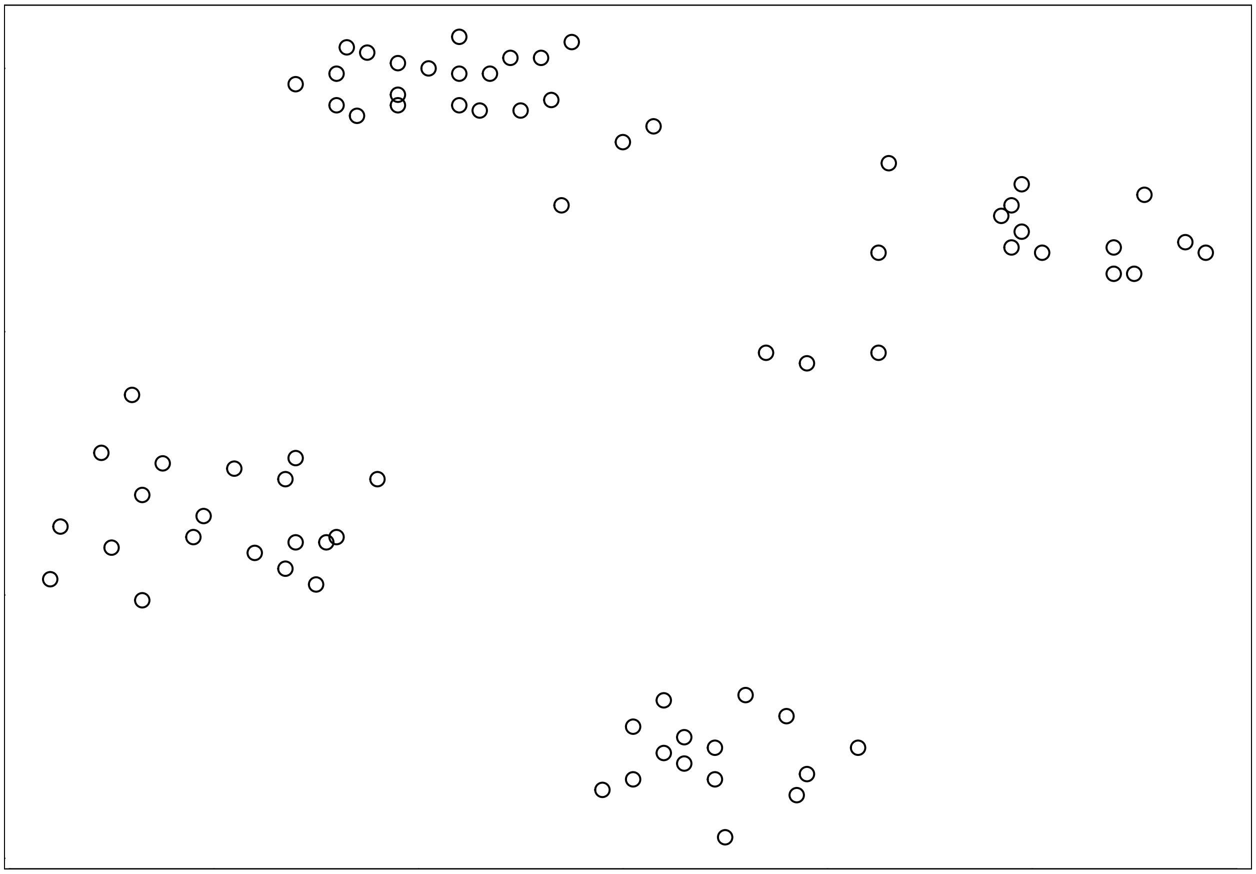
```
library(cluster)
data(ruspini)
head(ruspini)
```

```
##    x  y
## 1  4 53
## 2  5 63
## 3 10 59
## 4  9 77
## 5 13 49
## 6 13 69
```

```
summary(ruspini)
```

```
##        x                y
##  Min.   :  4.00   Min.   :  4.00
##  1st Qu.: 31.50   1st Qu.: 56.50
##  Median : 52.00   Median : 96.00
##  Mean   : 54.88   Mean   : 92.03
##  3rd Qu.: 76.50   3rd Qu.:141.50
##  Max.   :117.00   Max.   :156.00
```

```
par(mar=c(0,0,0,0))
plot(ruspini)
```



```
# See also data sets: flower, plantTraits, animals, iris, mtcars, etc.
data(iris)
dim(iris)
```

```
## [1] 150   5
```

```
summary(iris)
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width   
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100  
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300  
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300  
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199  
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800  
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500  
##        Species  
##  setosa    :50  
##  versicolor:50  
##  virginica :50  
##                 
##                 
## 
```

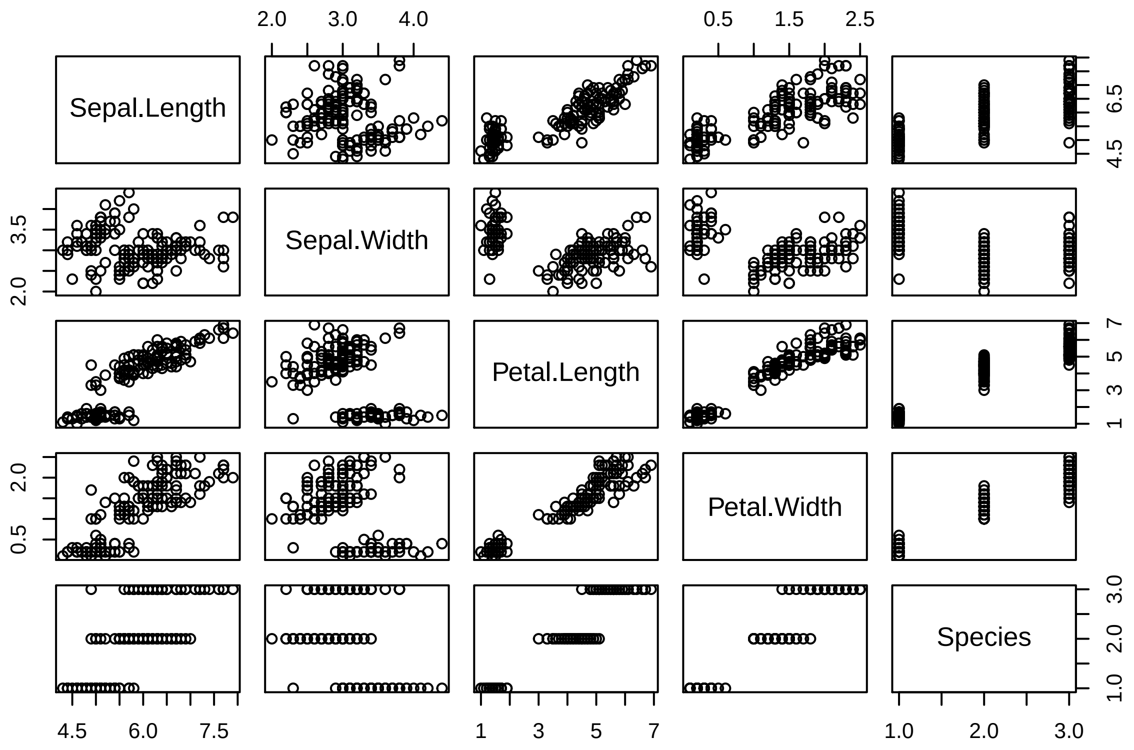
```
head(iris) # First six rows of a data table
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
tail(iris) # Last six rows of a data table
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
## 145          6.7         3.3          5.7         2.5 virginica
## 146          6.7         3.0          5.2         2.3 virginica
## 147          6.3         2.5          5.0         1.9 virginica
## 148          6.5         3.0          5.2         2.0 virginica
## 149          6.2         3.4          5.4         2.3 virginica
## 150          5.9         3.0          5.1         1.8 virginica
```

```
pairs(iris)
```



Challenge yourself

Again, remember: these sections are not graded. They are for you to explore the topic further.

1. Create a new R Markdown document or by opening the template we prepared for you. Go to File -> New File -> R Markdown…. or File -> Open -> M Markdown Template. In "Title" please put "Homework 1," in "Author" - your name and group number. Select PDF as a default output format.
2. Data is in the file carprise.csv. Read the data into R by using the read.csv command. Hint: to simplify the task, save the data file in your working directory, or change your working directory to where you store the file.
3. Create a vector x, which contains second column of the data matrix.
4. Create a vector y, which contains the fourth column of the data matrix.
5. Estimate a linear model by regressing y on x (hint: command is lm(formula = y ~ x))
6. Generate a summary of your linear model (command summary(model_name), where model_name is any name you've given your model (you have to use the name you have assigned your command to).
7. For the same model, create ANOVA output.

What do you observe? If you are struggling a bit here, refer to the "Contemporary Data Analysis" course, "Inferential Analytics" lecture. In it, I provide an elaborate explanation of the linear model and the ANOVA output.