```
################################################################################
#Introduction to Network Analysis in R
#
#In this half of the seminar we will learn the very basics of doing network analysis in R.
#We will begin by getting familiar with R as a programming platform. We will then
#learn how to work with network data in R. We will learn some of the functionality
#of igraph, a commonly used package for network analysis. We will learn how to
#plot a nice looking figure and do some simple calculations. This should
#serve as a basis for doing more advanced analyses in the future.

################################################################################

#Outline for R workshop

#Part I. Installing R
#Part II. R Basics: Running Commands in R, Comments, Help Files, Packages
#Part III. Objects and Operations
#Part IV: Network Data in R
#Part V: Plotting a Network in R
#Part VI: Simple Network Measures


#First, some helpful resources.
#The following will prove useful in the
#(unlikely) event that you get stuck or confused
#http://www.r-project.org
#http://cran.r-project.org/doc/manuals/R-intro.html
#http://www.luchsinger-mathematics.ch/Bashir.pdf #an intro guide

#There are many other resources on the internet
#just google "R introduction"


#############################################################################
#Part I: Getting R

#A. To download R (if you have not already done so):

#1. Go to: http://cran.r-project.org/
#2. Select mirrors on the left-hand menu.
#3. Select the location nearest you.
#4. Select the link for your operating system in the "Download and Install R" section.
#5. Make sure to download the base
#6. Follow the instructions for download

#############################################################################
```

#Part II. R Basics: Running Commands in R, Comments, Help Files, packages

#A. Commands in R

#Can run command on the command line
#Can run commands in R editor
#Can run commands from text editor

#B. Comments
#comments in R are handled using #
#any line with a # in front of it is not run

#C. Getting Help

#You can get information about functions
#(such as their inputs, outputs, and purpose) by using the ? command

#Help files are important as they describe the arguments
#to the functions as well the values that result from running the function

#For example, say we wanted to sum up a bunch of numbers and wanted learn how to do it

?sum

help.search("sum")   #search for a function if we are not sure of the name

#D. Packages
#Packages are sets of useful functions
#that researchers have contributed to the R community
#there are packages for nearly anything you would want to do
#they greatly enhance the functionality and appeal of R

#the only catch is that we need to install and load the packages
#as they are not immediately available when you open R
#you only need to install a package once (so it is saved on your computer)
#but you will have to load the package every time you restart R (assuming that
#package is necessary for that session)

#installing packages
#here will install one package that will be useful for network analys: igraph
#network and sna are other packages for doing network analysis
#note there are other packages that run more sophisticated statistical
#models, such as: ergm,tergm RSiena, latentnet

```
#The function to install packages is install.packages. The basic input is the name of the package,
#note that you put the name of the desired package in quotes
install.packages("igraph")


#We load packages by using a library command. The input is the name of the package, not in
quotes

library(igraph) #load igraph

#by loading the igraph package we can use all of the functions
#in that package

#We can look up all the functions within a package:

help(package=igraph)
#This shows us the range of things one could do using that particular package

#we can detach a package if we no longer want it loaded
#this is sometimes useful if two packages do not play well together
#for example to detach igraph we would type:
#detach(package:igraph)

################################################################################
###


#Part III. Objects and Operations
#the logic of R is based on objects, which can take many forms,
#and operations on those objects, this is somewhat different from
#statistical programs like sas or stata which are based on
#the dataset as the primary object of interest
#Here, we offer a very brief tutorial on different types of objects in R, beginning with scalars.

#A. Scalars
#we start with simple operations on a single number (or scalar)

a<-20  #assigning
a=20   #same as above


a      # looking at our object
#note that if we do not assign our output a name we cannot use it later

a+10    #basic math
```

```
#here a+10 is not assigned to anything

a/10

sqrt(a)

d=sqrt(a)

d #taking a look at result of operation
ls()   #list all objects, note there is nothing corresponding to a+10


d==a   #logicals, is d equal to a

d!=a #is d not equal to a?

d<a    #is d less than a?

d>a #is d greater than a?

rm(d)   #remove an object

ls() #it's gone!


#B. Vectors
#R also makes it easy to create and manipulate vectors

v=c(2,3,4) #the c command combines the input arguments
v

v[2]       #selecting the second element of the vector

v2=c(v,v)  #you can create a longer vector from two smaller vectors
v2

v3=c(2,3,"apple")  #mixing character with numerics makes it all character in a vector
v3
#we can use the class function to see what type of object we have
class(v3)
class(v2)

a=c(2,4,3,5,3)

a*3          #operations on a vector, elementwise
```

```
a+1  #add 1 to each element of a

a+a  #adds the nth element of a to the nth element of a

a==3        #logicals, is each element of a equal to 3?

sum(a==3)     #how many =3?
a[a==3]       #subset the vector, only keep elements that equal 3
length(a[a==3]) #subset the vector, only keep elements that equal 3, then ask how long it is
#note the length command returns the number of elements in the object

#C.Matrices

mat=matrix(1:36, nrow=6,ncol=6)  #creating a 6 by 6 matrix
#nrow specifies number of rows
#ncol specifies number of columns

mat

mat[1,]               #getting the first row

mat[,1]               #getting the first column

mat[2,3]              #getting the element in the 2nd row and 3rd column

mat[1:3,]             #getting the 1st through 3rd rows

mat[1:3,1:3]          #getting the 1st through 3rd rows   and the 1st through 3rd columns

mat2=cbind(1:10,21:30)  #can also make matrices using cbind or the rbind command
mat2                  #cbind=puts together as columns

mat3=rbind(1:10,21:30)  #rbind puts together as rows
mat3

dim(mat2)                 #check the number of rows and columns using dim
nrow(mat2)                #check the number of rows using nrow
ncol(mat2)      #check the number of columns using ncol

#for dim, nrow, or ncol, input is the matrix of interest

mat4=mat2*2    #can operate on the matrix elementwise, here multiply each element by 2
mat4

mat2+mat4      #here add two matrices together
```

```
mat2>4        #logicals, which elements are greater than 4?

mat2%*%mat3    #matrix multiplication
#key difference is the %*% to signify matrix multiplication

#D. Lists

#lists are a helpful way to store data
#anything can put into a list: vectors, matrices, characters.
#networks themselves are stored as lists in many packages

alist=list(1:6,mat2,"LINKS")
alist

alist[[1]]    #pulling the first part of the list, note the double bracket
alist[[2]]
alist[[3]]

alist[[1]][5]   #get the 5th element in the first slice of the list

alist=list(slice1=1:6,slice2=mat2)  #putting names on the list
alist

alist$slice1   #using the names to grab the first slice of the list


#E. Data Frames
#Data frames are helpful as you can put character and numeric together (unlike matrices)
#first, let's create little toy dataframe to work with

example_data=data.frame(
  grade=c(12,12,10,11,9,10),
  race=c("Hispanic","Hispanic","White","White","Native American","Black"),
  gpa=c(3,1.8,2.5,3.5,2.4,3.0))

#note that it goes column-wise, like cbind
#the function is data.frame, the inputs are the columns that you want turned into a dataframe

example_data

example_data[,1] #grabbing first column
example_data[,"grade"] #grabbing first column, same as above but using variable name

example_data$grade #does same thing as above

example_data[1,] #grabbing first row
```

```
#we can use the class function to see what type of variable we have
class(example_data[,2])   #note that our race variable is a factor
example_data[,2] #factors are like characters but they have an order and
#and a set number of values the variable can take

example_data[,2]=as.character(example_data[,2])  #turning race  into a character variable using
as.character
example_data[,2]
class(example_data[,2])



#########################################################################

#saving and loading objects into R
#note that we can save all or specific objects from our workspace using
#a save command
save(example_data, mat, file="") #you list the things you want saved and then where to save it
#use save.image to save every object in the workspace
#you would then load that back in later on using a load command
load(file="")
#input is simply the location of the file of interest

#########################################################################



#Part IV: Network Data in R

#A. Matrices as Inputs

#The first step in analyzing network data in R is to read the
#data into R and then to construct a network from that raw data

#This example will use network data based on classroom interactions
#from the work of Dan Mcfarland

#the actors are students in a classroom.
#the survey asked students to nominate classmates that they “hung around” with as friends."
#we also have individual characteristics on gender, race and grade

#the network data, who they hung around with as friends,
#is housed as a matrix. We also consider alternative representations below.

#The matrix is on the course website. The data is stored as class555_matrix.csv. We will read in
```

the matrix directly from the website using a read.csv function.

#Let's go ahead read in the classroom network in the matrix format. read.csv is the function, the inputs are the location of the file of interest:

```
class_mat=read.csv(file="https://sites.google.com/site/jeffreysmithdatafiles/class555_matrix.csv"
)

class_mat
#note that a 1 means that i nominates j as a friend; 0 means
#i did not nominate j as a friend

#now let's turn that into a matrix
#The function to turn a dataframe into a matrix is as.matrix
#The input is the dataframe of interest

class_mat=as.matrix(class_mat)

#let's put some rownames on the matrix, same as the column names
rownames(class_mat)=colnames(class_mat)

class_mat
```

#Let's also read in the attribute data, called: class555_attributedata.csv. Again, we read in the data directly using a URL:

```
class_attributes=
  read.csv(file="https://sites.google.com/site/jeffreysmithdatafiles/class555_attributedata.csv")
class_attributes
```

#This is a simple dataset describing the gender, grade and race of each student in the network. Note that the order of the dataframe has to be the same as the order in the matrix (and has to match the ids in the edgelist). Note also that the first column is the id of each actor in the network.

```
#We can grab particular columns by using a $ command
class_attributes$gender

#or by calling a particular column:
class_attributes[,"gender"]

#Note that there are multiple packages in R that perform network
#analysis, the main ones are sna and igraph
#sna goes along with packages ergm, network, tergm, latentnet
#while igraph is distinct from the other network packages
```

#we will go over how to construct networks using igraph

#A. Using matrices to construct a network using igraph

library(igraph) #loading igraph

#Let's go ahead and start by constructing a network using the matrix as input. The function is graph_from_adjacency_matrix. The inputs are: adjmatrix=the input matrix, mode= directed or undirected.

#Here we will create a network object called class_netbymatrix using the class_mat as the input.
class_netbymatrix=graph_from_adjacency_matrix(adjmatrix=class_mat,
                                              mode="directed")
class_netbymatrix

#Now, let's map the attributes onto that network object. Here we need to use a set_vertex_attr function. This makes it possible to take an attribute, like gender, and map it to the nodes in the network.

#The inputs are: graph=network of interest; name=name of variable on network to be created; value=a vector of attributes

#Let's first do gender, adding an attribute called "gender" to the network, equal to the gender values in class_attributes$gender we will map gender to the network created above,
class_netbymatrix

class_netbymatrix=set_vertex_attr(graph=class_netbymatrix,name="gender",
                                  value=class_attributes$gender)

#Now grade:
class_netbymatrix=set_vertex_attr(graph=class_netbymatrix,name="grade",
                                  value=class_attributes$grade)

#Now race:
class_netbymatrix=set_vertex_attr(graph=class_netbymatrix,name="race",
                                  value=class_attributes$race)

#Note that we can name the attributes anything we like using the name input, and it need not match the variable names on the original dataset.

class_netbymatrix
#We can see that we have now added gender, grade and race to the network

#At this point we have the basic network object constructed and can start to analyze, plot, etc.

#B. Edgelist as inputs

#alternatively, we can represent our network data as an edgelist
#this is often preferable to a matrix as it only requires that one
#records the edges, or ties, and thus we can ignore all of the 0s
#this makes them good options for very large networks, where
#it will be difficult to hold the matrix

#the basic form is for the first column to be the sender of the tie
#and the second column is the receivier of that tie
#it holds all of the information of the matrix, except that it will
#not show if there are isolates as they will not receive or send out any ties

#the edgelist is also saved on my website. first go ahead and download it and take a look
#the edgelist is called: class555_edgelist.csv

#now let's read in the edgelist
class_edges=read.csv(file="https://sites.google.com/site/jeffreysmithdatafiles/class555_edgelist.csv")
class_edges

#With an edgelist, we will want to use a graph_from_data_frame statement to construct our network
#the inputs are:
d=the edgelist
#directed= whether it is directed or not (T/F)

class_netbyedgelist=graph_from_data_frame(d=class_edges,directed=T)

#One advantage of using an edgelist is that it is easy to incorporate attributes into the network object. This simply requires using a vertices option when specifying the graph_data_frame statement. The input to vertices is a dataframe of the attributes of each actor, here class_attributes. igraph will add every variable in the dataset (except the first column, which is assumed to be the ids of the actors) to the created network object.

#Here we will go ahead and redo the statement above using a vertices option.
class_netbyedgelist=graph_from_data_frame(d=class_edges,directed=T,vertices=class_attributes)
class_netbyedgelist

#note that using an edgelist may lead to unintended results if there are isolates
#as they will not be listed on the edgelist!

```
#The edgelist is thus (often) insufficient to construct the network by itself
#we can make sure that we get the right number of nodes by including a vertices option
#showing the characteristics of all nodes (including isolates) as a dataframe
#here I include a data frame containing  a simple id variable,
#1 to 24 for each node in the network and include that as a input
#graph_from_data_frame(d=class_edges,directed=T,vertices=data.frame(id=1:24))


#########################################################################
#Part V: Basic Plotting of Network Data in R

#We will often start our analysis by plotting our network. this will give us
#a starting point for trying to understand what the social system looks like.
#It offers a much more intuitive representation than the matrix or edgelist
#In fact, one of the real benefits of a network approach is that there is a tight connection
#between the underlying data and the visualization of that data.

plot(class_netbymatrix) #plot will make a default picture of the network
plot(class_netbymatrix,layout=layout_in_circle) #same basic plot but
#here we will use a different layout, a circle layout


#let's color the nodes by grade
#we can do this by using a V(g)$color command
#we just need to set the color as the colors that we want, here based on grade

V(class_netbymatrix)$color=vertex_attr(graph=class_netbymatrix,"grade")
#the inputs for the V() function are the network of interest
#you set that equal to what colors you want different nodes to take
#here grabbing the grade of each student and using that differentiate the colors

plot(class_netbymatrix) #now, just plot as before

#and now gender
# but this time let's control the colors a bit more
#let's make our boys blue and girls pink

cols=ifelse(class_attributes$gender=="Female","pink","blue") #using an ifelse statement
#to make color blue or pink, pink if equals Female, blue otherwise
cols

table(cols,class_attributes$gender)

V(class_netbymatrix)$color=cols #now making color on network correspond to gender
#but based on specific colors of our choosing
```

```
plot(class_netbymatrix) # plot as before

#could also specify within
#the function itself using a vertex.color option
#plot(class_netbymatrix,vertex.color=cols)

#let's take out those labels and make the nodes a little smaller
plot(class_netbymatrix, vertex.label=NA,vertex.size=5)
#vertex.label controls the labels, vertex.size controls the size of the nodes

#we can also control the size of the nodes more carefully

#let's scale the nodes by their indegree, how many ties people send to them
#we can calculate this directly on the matrix using a colSums command:

indeg=colSums(class_mat) #this simply sums of the columns, showing how many
#ties each person recieved
indeg

#now let's scale the size by indegree
plot(class_netbymatrix,vertex.color=cols, vertex.label=NA,vertex.size=(indeg))
#vertex.size controls the size of the nodes

plot(class_netbymatrix,vertex.color=cols, vertex.label=NA,vertex.size=indeg+3)
#here all the nodes are little bigger (adding a 3 to indegree) but still sized by indegree

#now let's make the arrows a little smaller
plot(class_netbymatrix,
vertex.label=NA,vertex.size=indeg+3,edge.arrow.size=.5,edge.arrow.width=.5)
#edge.arrow.size and edge.arrow.width control the arrows

#changing the color of the lines
plot(class_netbymatrix, vertex.label=NA,
    vertex.size=indeg+3,edge.arrow.size=.5,edge.arrow.width=.5,edge.color="light gray")

#edge.color controls the color of the edges

#here will add a bit of curve to the edges for a nice effect
plot(class_netbymatrix, vertex.label=NA,
    vertex.size=indeg+3,edge.arrow.size=.5,edge.arrow.width=.5,edge.color="light gray",
    edge.curved=T) #if we want to add curve to the edges add edge.curved=T

plot(class_netbymatrix, vertex.label=NA,
    vertex.size=indeg+3,edge.arrow.size=.5,edge.arrow.width=.5,edge.color="light gray",
    edge.curved=.3) #here we set edge.curved to .3, less than the default.
```

```
#one last option to show you, same thing as before but here we will
#add a vertex.fame.color option. by setting that to NA, we take off the
#black edge to the nodes
plot(class_netbymatrix, vertex.label=NA,
    vertex.size=indeg+3,edge.arrow.size=.5,edge.arrow.width=.5,edge.color="light gray",
edge.curved=.3,vertex.frame.color=NA)

#look at the following help files for more options
?plot.igraph
?igraph.plotting

################################################################################
####

#Part VI: Simple Network Measures
#In this section we will walk through the calculation of a few example network measures. The
idea is to get a bit more facility with R and to lay some of the groundwork for more advanced
analyses. We will examine how to calculate these network measures using the matrix
representation of the network, as well as with functions within the igraph package (this can also
be done within the sna package).

#Degree
#We start with degree, which is a simple measure capturing the number of edges for each actor.
We can define outdegree as the number of edges sent from actor i and indegree as the number of
edges received by actor i. We start by calculating degree using the raw matrix, class_mat.

#We calculate outdegree by summing over the rows of the matrix. Each element of the matrix
shows if i sends a tie to j. By summing over the rows, we calculate the total number of edges sent
by i.
outdeg=rowSums(class_mat)
outdeg #We see that the first actor nominates 4 friends, the second actor nominates 2 friends and
so on.

#We calculate indegree by summing over the columns of the matrix, showing the total number of
edges received by i.
indeg=colSums(class_mat)
indeg #We see that the first actor receives 1 nomination (for example).

#The function is degree. The main inputs are:
#graph=network
#mode=type of degree calculation: in, out, or total (adding up outdegree and indegree).
outdeg_igraph = degree(graph=class_netbymatrix, mode="out")
indeg_igraph = degree(graph=class_netbymatrix, mode="in")

outdeg==outdeg_igraph #the same!
```

#Density
#Density is another simple measures of network structure that captures the total number of ties in the network divided by the total number of ties possible.
#Let's first calculate the density "by hand" using the size and number of edges in the network.

num_edges=gsize(class_netbymatrix) #function gsize get the number of edges
num_nodes=gorder(class_netbymatrix) #function gorder gets the number of nodes

num_edges
num_nodes
#We can also calculate the number of dyads (excluding ii pairs) which tells us how many ties are possible in the network.
number_dyads=(num_nodes*(num_nodes-1))

#Calculating density
den=num_edges/number_dyads
den

#now using canned function, edge_density, in igraph.
edge_density(class_netbymatrix) #the same!

#Walks
#We now turn to walks on the network, defined as any sequence of nodes and edges (backwards and forwards) that connect i to j. So, i->j->k->j->l would be a walk of length 4 from i to l. One item of interest is the number of walks of a given length between two nodes. We can use matrix multiplication to calculate this. By raising the matrix to the nth power, we get the number of walks of length n between all ij pairs.

#Let's calculate the number of walks of length two by multiplying the matrix by itself:
walks2=class_mat%*%class_mat
walks2
#This suggests, for example, that there is 1 walk of length 2 between actor 1 and actor 1 (themself). In this case, the walk is : 1->7->1. We can see this by looking at the rows for actor 1 and actor 7:

class_mat[c(1,7),]

#We can see that actor 1 is friends with 3, 5, 7 and 21. We can also see that actor 7 is friends with 1, 9, 10 and 16, thus creating a walk of length 2: 1->7->1.

#We can calculate the number of walks of length 3, 4, etc. in an analogous fashion. For example, for walks of length 3:
walks3=class_mat%*%class_mat%*%class_mat
walks3 #Here, we see that there are 0 walks of length 3 from 1 to 1, but 4 different walks from 1 to 3 (for example, 1->3->8->3).

#Paths, Distance, and Closeness
#Paths are defined as a sequence of nodes and edges starting with one node and ending with another, where a path is not allowed to revisit the same node twice (unlike with walks). For example, i->j->l would be a path of length 2. It is often of interest to know the shortest path, or the distance, between nodes in a network. Here we will rely on the distances function within the igraph package.

#The main inputs are:
#graph=the network
#mode=type of distance to calculate, where the 'out' option says we want distance from i to j

dist_mat=distances(graph=class_netbymatrix, mode="out")
dist_mat #This is a matrix holding the length of the shortest path between all pairs of nodes. We can see, for example, that actor 1 is 1 step from actor 3 and 2 steps from actor 6. Note that Inf means that i can not reach j through any path.

#We can get the specific paths connecting i to j using the all_shortest_paths function. The main inputs are the graph and the starting node (from) and the ending node (to). Here we see all the shortest paths between 1 and 6.

all_shortest_paths(class_netbymatrix, from=1, to=6) #If we look at the first part of the output, we can see the path between 1 and 6 (1->3->6).

#Or, what about the shortest paths between 1 and 16?
all_shortest_paths(class_netbymatrix, from=1, to=16) #We can see there are two paths of length 2 between 1 and 16 (1->7->16; 1->3->16).

#It is often of interest to summarize the distances over all ij pairs. For example, we may be interested in the mean distance between nodes. Let's go ahead and calculate the mean distance using our matrix calculated above. Before we calculate the mean, let's put an NA on the diagonal, as we are not interested in the distance to oneself (which is by definition 0).

diag(dist_mat)=NA
mean(dist_mat,na.rm=T) #Here we calculate the mean distance across all pairs, excluding the NAs on the diagonal.

#The mean is Inf, as there are Inf values (so i and j are not reachable) in the matrix. This is not especially informative, however, only telling us that at least one pair is unreachable. One option would be to exclude the unreachable pairs. This is a common approach but also throws out information on all cases where i cannot reach j.

mean(dist_mat[dist_mat!=Inf], na.rm=T) #Ignoring the unreachable pairs, we see a mean of 2.81, so that actors are, on average, separated by paths of length 2.81.

#Alternatively, researchers will often employ closeness as the summary measure when there are

unreachable pairs. Closeness is based on the inverse of the distance matrix. By inverting the distance matrix, all Inf values are turned into 0s and thus can be included in the mean calculation. The inverse of the distance matrix has the opposite interpretation as above, showing show how 'close' actor i is to actor j. The disadvantage of a closeness measure is that the interpretation is not as intuitive as with distance. Let's go ahead and calculate the mean closeness for this network.

close_mat=1/dist_mat #taking the inverse of the distance matrix
close_mat #We see NAs on the diagonal and the closeness values for other cells in the matrix. The values range from 0 (not reachable, so minimum closeness) to 1 (directly connected, so maximum closeness).

mean(close_mat, na.rm=T) #Here we calculate the mean closeness across all pairs, excluding the NAs on the diagonal. We see a value of .348. Actors are, on average, 1/2.87 paths close to each other.

#It may also be useful to calculate the median, as a means of comparison:
median(dist_mat, na.rm=T) #here for distance
median(close_mat, na.rm=T) #here for closeness

#In this case we can see that the median yields the same information in both cases, that the median distance is 3 and the median closeness is 1/3.

#Reachability
#Reachability captures whether actor i can reach actor j through any path. This can be calculated directly from the distance matrix. Actor i can reach actor j if the distance between i and j is less than Inf (i.e., there is some path between i and j).

#Here we use an ifelse function to set the reachability matrix to 1 if distance is less than Inf and 0 if it is not less than Inf.
reach_mat=ifelse(dist_mat <Inf, yes=1, no=0)
reach_mat #We can see that actor 1 cannot reach actor 2 (for example).

#Diameter
#We can also use the distance matrix to calculate diameter, showing the longest distance between any two nodes in the network.

#Note that if we simply calculate the maximum over the distance matrix we will get back Inf if there are nodes that cannot reach other:
max(dist_mat, na.rm=T) #calculating max distance, excluding NAs on the diagonal

#We may, instead, be interested in calculating the maximum distance just over those nodes that can reach other. Here, we exclude any distance that is Inf.
max(dist_mat[dist_mat!=Inf],na.rm=T)

#We can also calculate diameter using the diameter function in igraph. By default, the function

will include only those pairs that are reachable.
diameter(class_netbymatrix)