Andrew Sparkes - STAT332 Statistical Analysis of Networks Sec 01 - Lab 1

STAT 332: Lab 1

1.) Setup

Welcome to Network Analysis in R. R is a statistical programming language with a vast history of user developed packages that we are able to use in order to analyze data. In our case, we will be concerned with loading a few packages that are critical to our work in network analysis and then using these packages to analyze networks.

Our integrated developer environment is called R Studio (though, you are allowed to use Positron if you would rather). I'll demonstrate in class how to create an R environment for your work in this class. The steps are rather easy:

1.) Under File select New Project

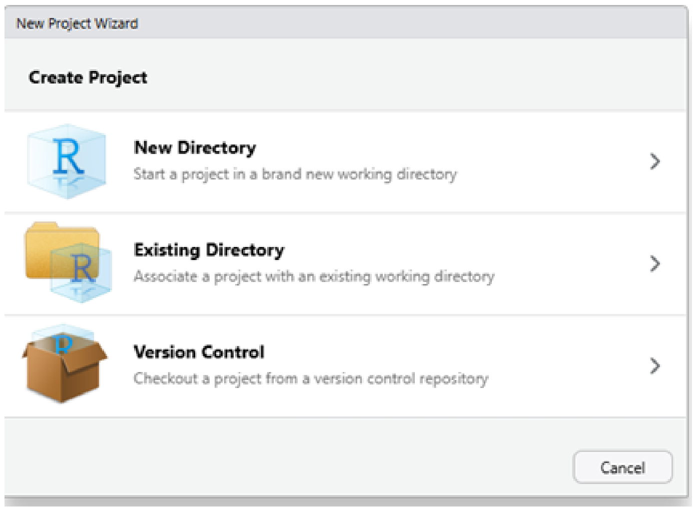

New Project Wizard
Create Project
New Directory
Start a project in a brand new working directory
Existing Directory
Associate a project with an existing working directory
Version Control
Checkout a project from a version control repository
Cancel

Figure 1: Image showing the new project dialog

2.) Select New Directory or Existing Directory (I would suggest creating a new directory)

3.) Select "New Project"

4.) Enter the name of your new directory (You can browse to make it a subdirectory).

Make sure that you select "Use renv with this project) which will create a new R environment for your project. You will want to load up this environment every time you work on your network data analysis. Creating a git repo for your work is optional.

When you open up the folder you starte the new project in, you will see something like this:

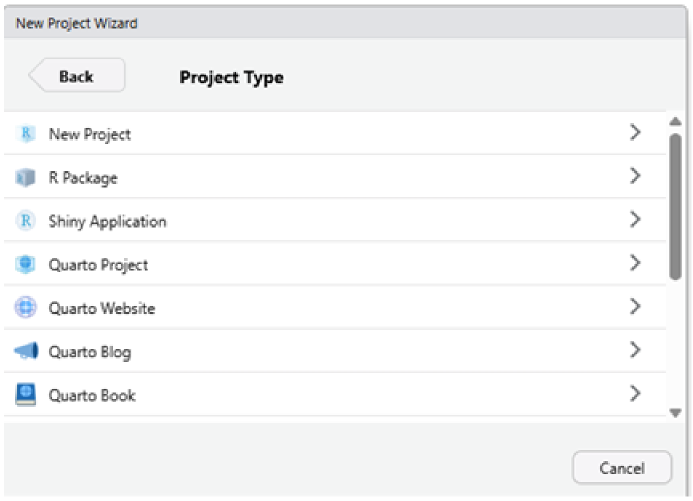
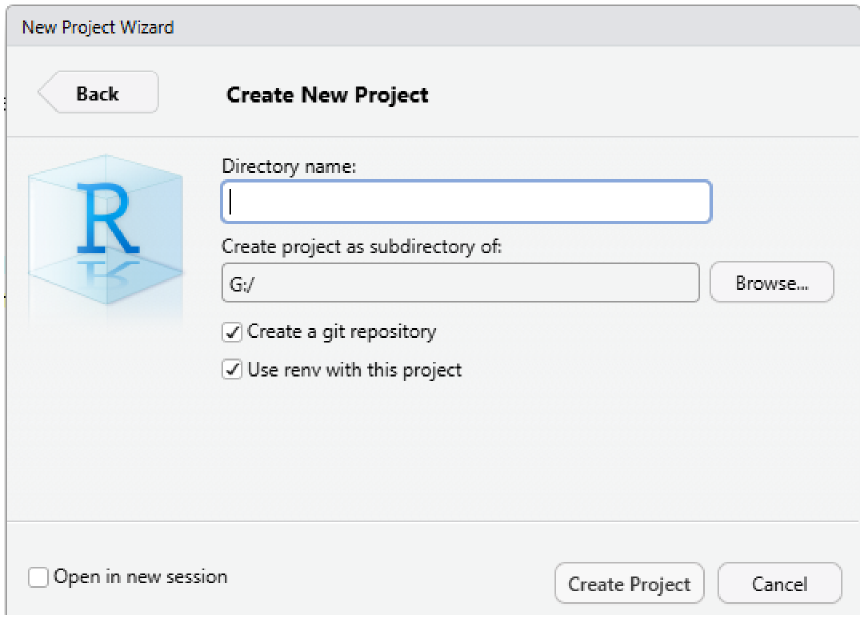


Figure 2: Image showing dialog for project type



Directory name:

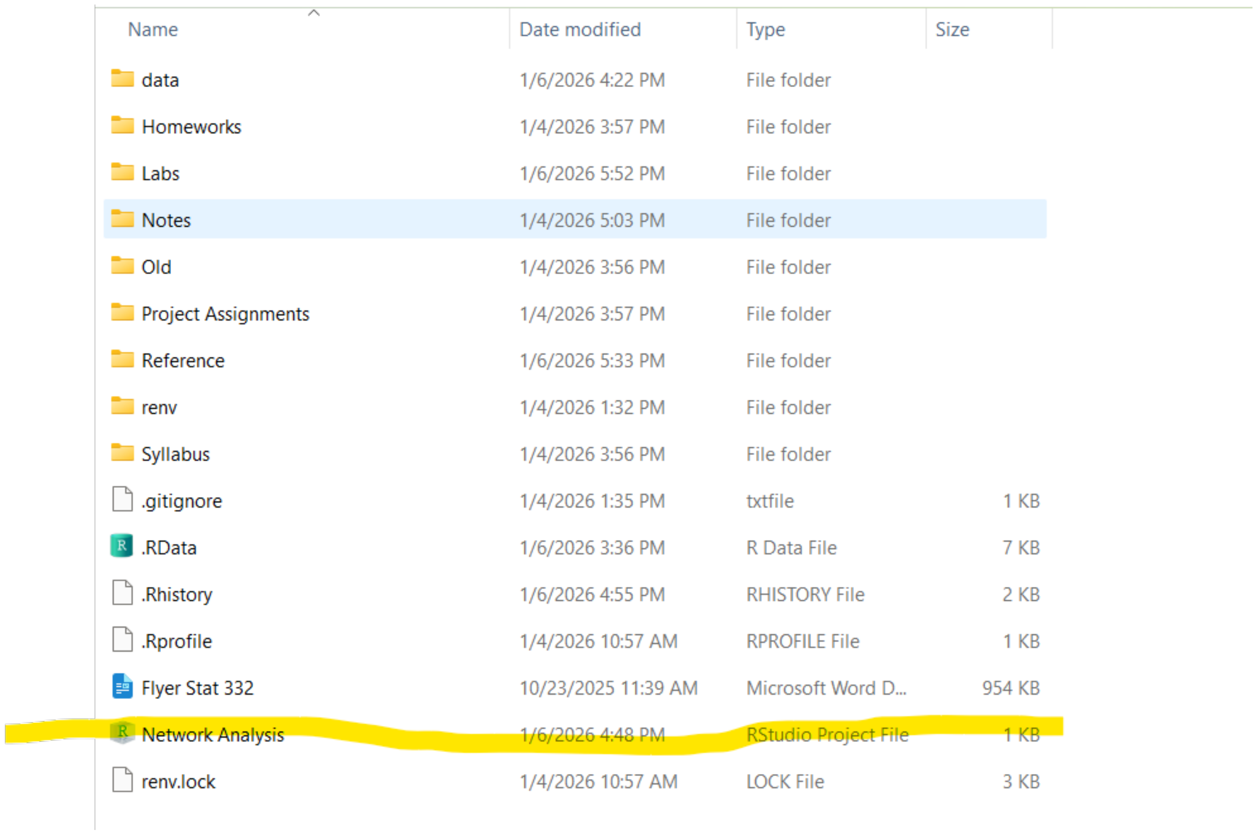
Create project as subdirectory of: G:/

[✓] Create a git repository

[✓] Use renv with this project

[ ] Open in new session

Figure 3: Image showing the "Create New Project" Dialog



Name	Date modified	Type	Size
data	1/6/2026 4:22 PM	File folder	
Homeworks	1/4/2026 3:57 PM	File folder	
Labs	1/6/2026 5:52 PM	File folder	
Notes	1/4/2026 5:03 PM	File folder	
Old	1/4/2026 3:56 PM	File folder	
Project Assignments	1/4/2026 3:57 PM	File folder	
Reference	1/6/2026 5:33 PM	File folder	
renv	1/4/2026 1:32 PM	File folder	
Syllabus	1/4/2026 3:56 PM	File folder	
.gitignore	1/4/2026 1:35 PM	txtfile	1 KB
.RData	1/6/2026 3:36 PM	R Data File	7 KB
.Rhistory	1/6/2026 4:55 PM	RHISTORY File	2 KB
.Rprofile	1/4/2026 10:57 AM	RPROFILE File	1 KB
Flyer Stat 332	10/23/2025 11:39 AM	Microsoft Word D...	954 KB
Network Analysis	1/6/2026 4:48 PM	RStudio Project File	1 KB
renv.lock	1/4/2026 10:57 AM	LOCK File	3 KB

Obviously, I have a lot more folders than you are likely to need, but some of these are auto-created. When you load R-Studio using the highlighted link, it will open your project which contains all of the libraries that you will download and use for your work in network analysis (these are located in the renv file). We will be installing packages through the package manager:

The install button on the package manager will give us a dialog box to type in where we will get our packages for this course.

Your first task will be to download some of the libraries we are going to be using. You can start with this list:

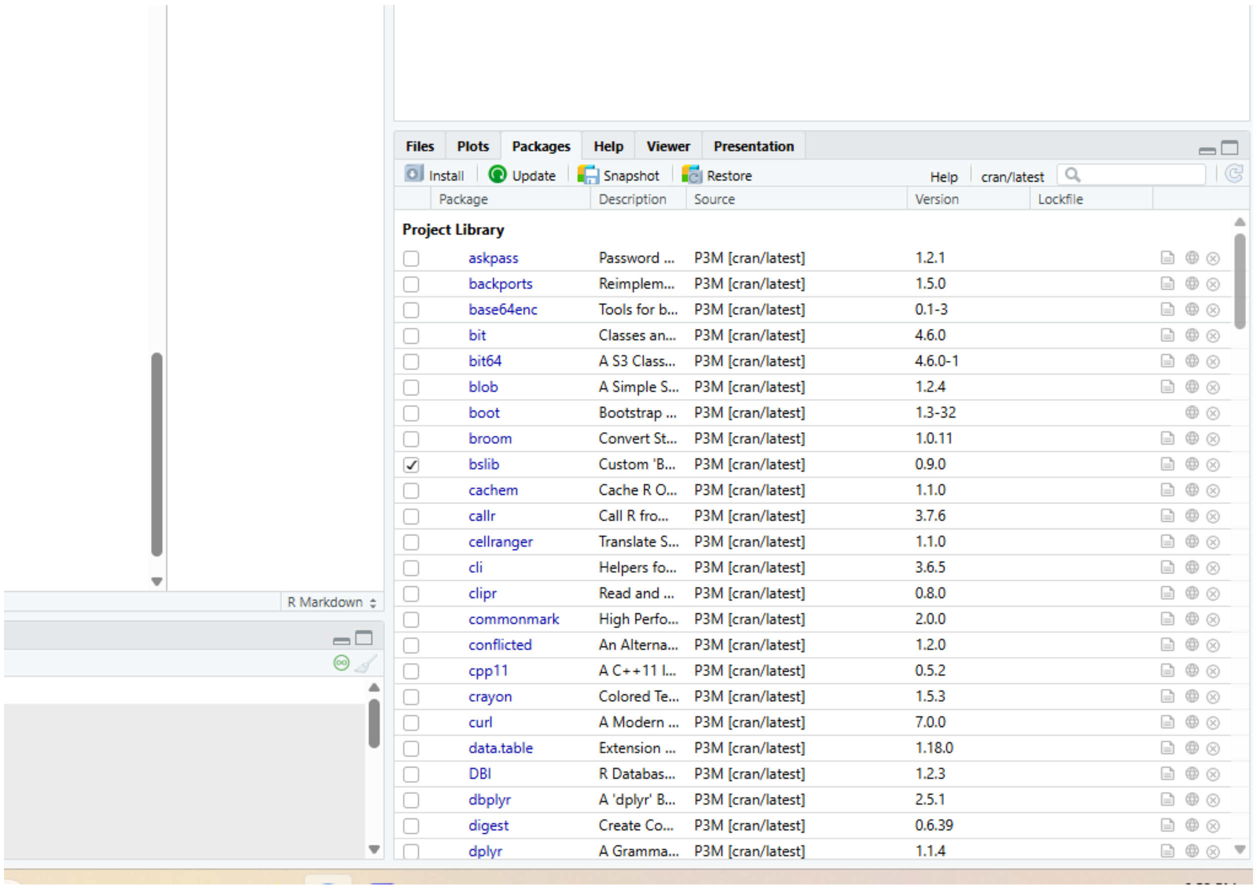
- igraph
- igraphdata

Once you have these installed, we will get started.

2.) Working with packages and Graph Data

The first thing I'd like you to do is to to create a new .Rmd file. This is called a markdown file. You can place it wherever you would like within the folder; however, you need to create this file using File-> New File -> R Markdown once you have opened R Studio by the link created for your project. You should get something that looks like this:

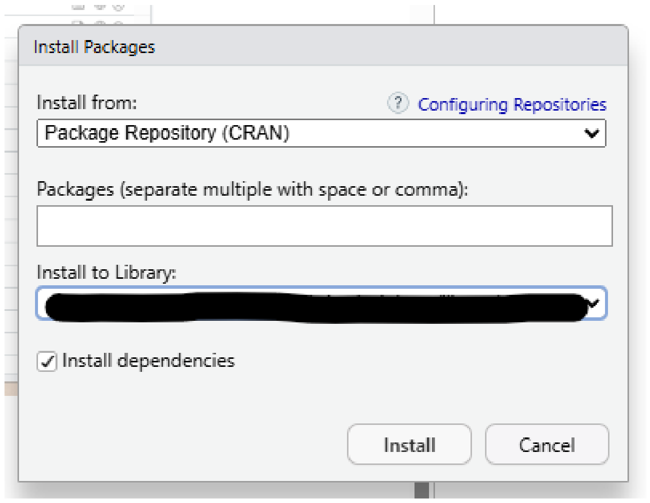
Notice that I have clicked the "Visual" tab so that I can write in a format that's similar to a typical document editor. The title: and output: sections are part of something called the YAML. This will give us the output type for the file which is currently html. In future labs and homework you will need to know how to create these files, so save this for your documentation.



Project Library

	Package	Description	Source	Version	Lockfile
	askpass	Password ...	P3M [cran/latest]	1.2.1	
	backports	Reimplem...	P3M [cran/latest]	1.5.0	
	base64enc	Tools for b...	P3M [cran/latest]	0.1-3	
	bit	Classes an...	P3M [cran/latest]	4.6.0	
	bit64	A S3 Class...	P3M [cran/latest]	4.6.0-1	
	blob	A Simple S...	P3M [cran/latest]	1.2.4	
	boot	Bootstrap ...	P3M [cran/latest]	1.3-32	
	broom	Convert St...	P3M [cran/latest]	1.0.11	
✓	bslib	Custom 'B...	P3M [cran/latest]	0.9.0	
	cachem	Cache R O...	P3M [cran/latest]	1.1.0	
	callr	Call R fro...	P3M [cran/latest]	3.7.6	
	cellranger	Translate S...	P3M [cran/latest]	1.1.0	
	cli	Helpers fo...	P3M [cran/latest]	3.6.5	
	clipr	Read and ...	P3M [cran/latest]	0.8.0	
	commonmark	High Perfo...	P3M [cran/latest]	2.0.0	
	conflicted	An Alterna...	P3M [cran/latest]	1.2.0	
	cpp11	A C++11 I...	P3M [cran/latest]	0.5.2	
	crayon	Colored Te...	P3M [cran/latest]	1.5.3	
	curl	A Modern ...	P3M [cran/latest]	7.0.0	
	data.table	Extension ...	P3M [cran/latest]	1.18.0	
	DBI	R Databas...	P3M [cran/latest]	1.2.3	
	dbplyr	A 'dplyr' B...	P3M [cran/latest]	2.5.1	
	digest	Create Co...	P3M [cran/latest]	0.6.39	
	dplyr	A Gramma...	P3M [cran/latest]	1.1.4	

Figure 4: Image showing the package manager in R Studio



Install Packages

Install from: Configuring Repositories

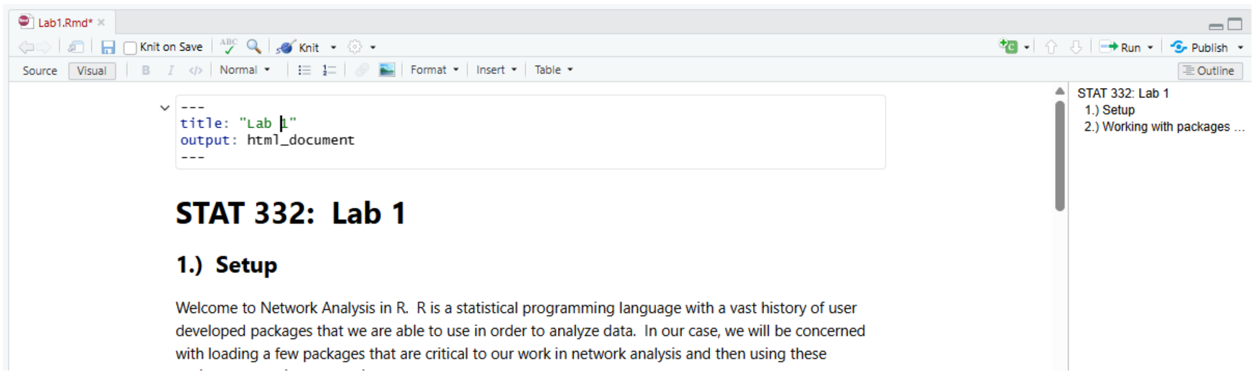
Package Repository (CRAN)

Packages (separate multiple with space or comma):

Install to Library: [redacted]

✓ Install dependencies

Figure 5: Package Manager Dialog



```
---
title: "Lab 1"
output: html_document
---
```

STAT 332: Lab 1

1.) Setup

Welcome to Network Analysis in R. R is a statistical programming language with a vast history of user developed packages that we are able to use in order to analyze data. In our case, we will be concerned with loading a few packages that are critical to our work in network analysis and then using these

Figure 6: Layout of the visual editor within R Studio

2.a) Getting started.

Using ctrl-alt-i you can create a section where you are going to embed R code. We're going to start by loading in the igraphdata library and exploring it a little bit.

```r
library(igraphdata)
```

```r
data(package = "igraphdata")
```

By running the library(igraphdata) command, it will load the igraphdata package into memory (RAM). The code-chunk of data(package = "igraphdata") opens a viewer that shows the data sets that are within this package. These are all graph-object data (igraph object) that we are able to use. Let's start with the most historical (and simplest) dataset, the Bridges of Koenigsberg data.

```r
data("Koenigsberg")
g <- Koenigsberg
rm(Koenigsberg)
```

After running this code cell, you will have done three things. First, you will have attached the Koenigsberg data. Second, you will create a copy of this data and called it g (because I'm too lazy to keep typing Koenigsberg every time). And finally, you will have removed the dataset Koenigsberg from your active memory. You can now see that the Data window in R Studio now includes the Koenigsberg data called g. Technically, this object is of type "list" however, the class of this object is "igraph". You can verify this:

```r
print(typeof(g))
```

```
## [1] "list"
```

```r
print(class(g))
```

```
## [1] "igraph"
```

So, it looks like it's time to learn "igraph". We need to load the library:

```r
library(igraph)
```

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

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

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

```
##     union
```

Now, let's look at this "g" object.

```
g
```

```
## This graph was created by an old(er) igraph version.
## i Call `igraph::upgrade_graph()` on it to use with the current igraph version.
## For now we convert it on the fly...

## IGRAPH 227bd5e UN-- 4 7 -- The seven bidges of Koenigsberg
## + attr: name (g/c), name (v/c), Euler_letter (v/c), Euler_letter (e/c),
## | name (e/c)
## + edges from 227bd5e (vertex names):
## [1] Altstadt-Loebenicht--Kneiphof
## [2] Altstadt-Loebenicht--Kneiphof
## [3] Altstadt-Loebenicht--Lomse
## [4] Kneiphof           --Lomse
## [5] Vorstadt-Haberberg --Lomse
## [6] Kneiphof           --Vorstadt-Haberberg
## [7] Kneiphof           --Vorstadt-Haberberg
```

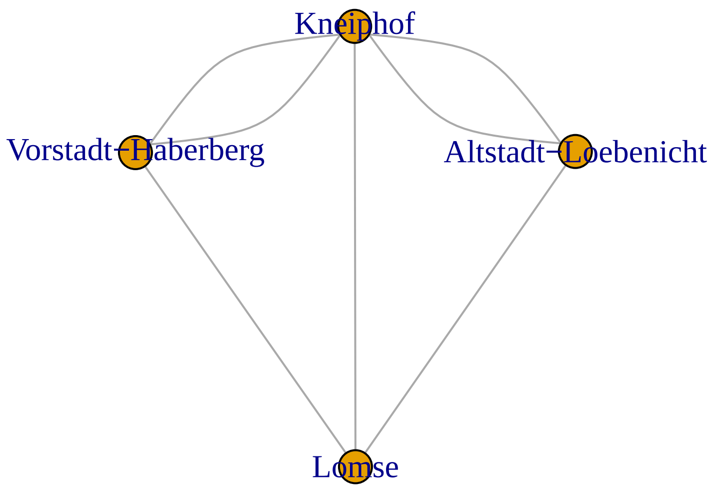
It's actually a good thing we started with a simple graph here, because just typing the name of the graph object will produce both a summary of the graph AND a list of all of the edges. If we just want the summary, we use the "summary" function.

```
summary(g)
```

```
## IGRAPH 227bd5e UN-- 4 7 -- The seven bidges of Koenigsberg
## + attr: name (g/c), name (v/c), Euler_letter (v/c), Euler_letter (e/c),
## | name (e/c)
```

Before we try to figure out what this output means, let's just get a quick plot of the data.

```
plot(g)
```



This graph data already exists, so we didn't have to create it ourselves, but we could have (and will need to) in the future. For now, we will continue exploring this data and then move on to data creation in part 3. At this time, we should probably parse out that summary function. Let's remind ourselves of the output:

```
summary(g)
```

```
## IGRAPH 227bd5e UN-- 4 7 -- The seven bidges of Koenigsberg
## + attr: name (g/c), name (v/c), Euler_letter (v/c), Euler_letter (e/c),
## | name (e/c)
```

Ignoring the numbers directly after IGRAPH, You can see that the ouput is: UN– 4 7 – The seven bridges of Koenigsberg. The first four characters indicate the poperties of the graph. U means unidrected and D means directed. N tells us that a vertex attribute called "name" exists (so that there are names for the verticies and numbers shouldn't be used for the labels). W would indicate that the graph is weighted. And B as the fourth character would indicate a vertex attributed called "type" (B is historic and represents a bipartite graph). Dashes indicate the absence of these parameters. The number 4 indicates that this graph has 4 vertices and the number 7 indicates the number of edges. The next two dashes are just to separate the title of the graph from the encoded information. The next thing that we can see is a list of the node and edge attributes of the graph. Anything with a "v" indicates that it's an attribute of the vertex and anything with an "e" indicates it's an edge attribute. Let's explore these attributes.

```
V(g)$name
```

```
## [1] "Altstadt-Loebenicht" "Kneiphof"            "Vorstadt-Haberberg"
## [4] "Lomse"
```

```
V(g)$Euler_letter
```

```
## [1] "B" "A" "C" "D"
```

```r
E(g)$Euler_letter
```

```
## [1] "a" "b" "f" "e" "g" "c" "d"
```

```r
E(g)$name
```

```
## [1] "Kraemer Bruecke" "Schmiedebruecke" "Holzbruecke"     "Honigbruecke"
## [5] "Hohe Bruecke"    "Gruene Bruecke"  "Koettelbruecke"
```

We're going to ignore the Euler letters as they aren't particularly important to this class (but I would strongly recommend looking up the history of this problem if you are interested!). V(g)$name gives us the names of the communities at each end of the bridges. These represent our vertex names for this graph. The E(g)$name gives us the names of the bridges ("Die Bruecke" means "The Bridge" in German, in case you were wondering.) There are no other edge properties to worry about here, so we are good to go.

So, from our notes, we can tell that this is an undirected graph. Create the adjacency matrix for this by hand before we look at how to produce it with code. For the record, adjacency matrix storage of graph objects is very memory inefficient, so I really wouldn't do this with a large graph.

```r
as_adjacency_matrix(g, sparse = FALSE)
```

```
##                      Altstadt-Loebenicht Kneiphof Vorstadt-Haberberg Lomse
## Altstadt-Loebenicht                    0        2                  0     1
## Kneiphof                               2        0                  2     1
## Vorstadt-Haberberg                     0        2                  0     1
## Lomse                                  1        1                  1     0
```

Notice that the diagonal is 0? What does this mean? Some of the nodes have multiple paths between them making this a multigraph. If we want to get the degree's of the nodes for this graph, we can do it using the "degree" function.

```r
degree(g)
```

```
## Altstadt-Loebenicht            Kneiphof  Vorstadt-Haberberg               Lomse
##                   3                   5                   3                   3
```

If you want to extract the edge list, you can do that with the E() function.

```r
E(g)
```

```
## + 7/7 edges from 227bd5e (vertex names):
## [1] Altstadt-Loebenicht--Kneiphof
## [2] Altstadt-Loebenicht--Kneiphof
## [3] Altstadt-Loebenicht--Lomse
## [4] Kneiphof           --Lomse
## [5] Vorstadt-Haberberg --Lomse
## [6] Kneiphof           --Vorstadt-Haberberg
## [7] Kneiphof           --Vorstadt-Haberberg
```

2b.) Statistics

Size Remember that the size of a graph is simply the number of vertices. This can be obtained from the summary function, but to isolate this number, we can use a function: gorder(). For what it's worth, another name for the size of a graph is the "order" of the graph; hence graph-order.

```r
gorder(g)
```

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

Components In order to count the number of components on a graph, we can do this with a function called: count_components()

```r
count_components(g)
```

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

Isolates Finding the number of isolates is actually a little bit weird. There is no built in function for this, but recalling that an isolate is a vertex with degree zero, we can use the degree function to produce a vector of the nodes and their degrees. What we need to do is to "count" the number of times where the degree is zero. This is done with the sum function. Hence:

```r
sum(degree(g) == 0)
```

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

If we want to look at the proportion of values that are isolates, we can do this using the gorder() function from earlier (yeah, this is the reason we needed that.)

Proportion of isolates:

```r
sum(degree(g) == 0) / gorder(g)
```

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

Density Let's use the function to find the density of the graph: edge_density()

```r
edge_density(g)
```

```
## [1] 1.166667
```

Oh no! Edge density can't be a number greater than 1! What could have caused this? If a graph contains loops or is a multigraph, the density calculation will be wrong. We need to adjust for this by looking at a graph where only the unique edges are present and loops are accounted for. The edge_density function can account for the loops using a bool edge_density(graph, loops = FALSE). Changing this to true for loops will adjust the calculation, but it still doesn't fix the other issue. You can actually perform both tasks at once using the "simplify" function.

```r
edge_density(simplify(g))
```

```
## [1] 0.8333333
```

```r

diameter(g)
```

Diameter

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

Average Path Length igraph used to have a function for average path length called: average.path.length(). However, this function is now depricated and has been replaced by the function: mean_distance()

```r
mean_distance(g)
```

```
## [1] 1.166667
```

2c.) Representation of summary statistics

Obviously, this is not very effective at presenting the results. We should make a summary table of these. I'm going to use the packages data.table and gt for this (if you don't have it, do you remember how to get it?)

ID	Value
Size	4.0000000
Components	1.0000000
Isolate Proportion	0.0000000
Density	0.8333333
Diameter	2.0000000
Average Path Length	1.1666667

```r
library(data.table)
table <- data.table(
  ID = c("Size", "Components", "Isolate Proportion", "Density", "Diameter", "Average Path Length"),
  Value = c(gorder(g), count_components(g), sum(degree(g) == 0) / gorder(g), edge_density(simplify(g)),
)
```

```r
library(gt)
table %>% gt()
```

3.) Your turn #1:

A collection of journal articles is provided for your consumption. I have provided csv files for both the nodes and links as well as code to read and create a network from this data. Using what you know from the code above, create a plot of this network and a table that summarizes the statistics describing this data.

```r
nodes <- read.csv("Citation_vertices.csv", header = T, as.is = T)
links <- read.csv("Citation_Edgelist.csv", header = T, as.is = T)
net <- graph_from_data_frame(d = links, vertices = nodes, directed = T)
```

Plot (remember, this is a directed graph)

```r
plot(net)
```

ID	Value
Size	46.00000000
Components	13.00000000
Isolate Proportion	0.19565217
Density	0.02222222
Diameter	8.00000000
Average Path Length	2.84883721

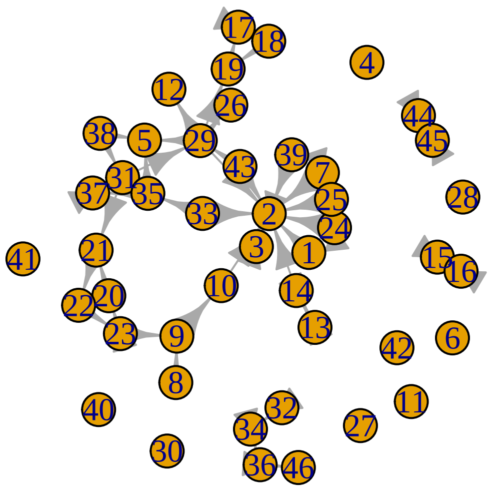


Table of Summary Information

```
make_table_happen <- function(graph) {
  data.table(
    ID = c("Size", "Components", "Isolate Proportion", "Density", "Diameter", "Average Path Length"),
    Value = c(gorder(graph), count_components(graph), sum(degree(graph) == 0) / gorder(graph), edge_dens
  ) |>
    gt()
}

make_table_happen(net)
```

ID	Value
Size	50.00000000
Components	5.00000000
Isolate Proportion	0.08000000
Density	0.04816327
Diameter	10.00000000
Average Path Length	4.32463768

4.) Your turn #2:

We are going to generate a random graph based on the Erdos-Renyi model. Your task is to produce the Plot and table of summary information for this network as well.

```r
set.seed(123)
g2 <- sample_gnp(50, 0.048)
```

Plot

```r
plot(g2)
```

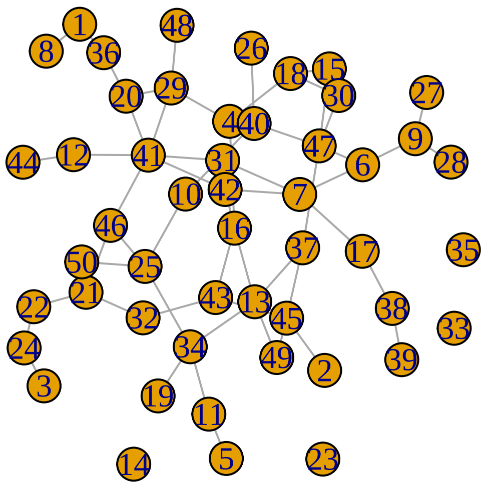


Table of Summary Information

```r
make_table_happen(g2)
```

5.) Results

Obviously, we have some work that needs done. Our plots don't look very good, we aren't representing any attributes of the vertices or edges, we aren't interpreting the statistical measures, we aren't finding clusters or cliques, we don't know how resistant these networks are, we don't know anything about the hubs or critical bridges in the data, etc…; however, we now have a start.

What you need to do nest is to "knit" your document into an html using the "Knit" button near the top of the screen. This will give you an HTML that you can turn in.