A First Introduction to Networks with R

Social Media and Web Analytics @ TiSEM

Lachlan Deer

Last updated: 11 April, 2021

What is a Network?

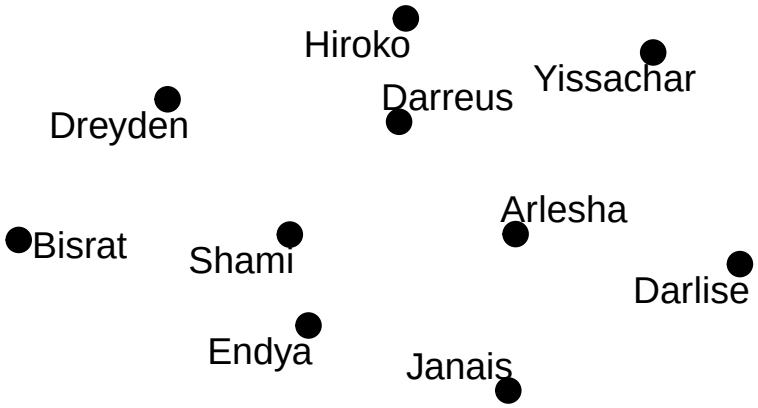
Loosely speaking, a network is a set of points connected by lines and/or arrows. The idea is that these points describe an entity and the lines between an entities represent the relationshis between them.

When you read about networks - some people (including me at times) are going to refer to them as "graphs". Think of "network" and "graph" as interchangeable terms.

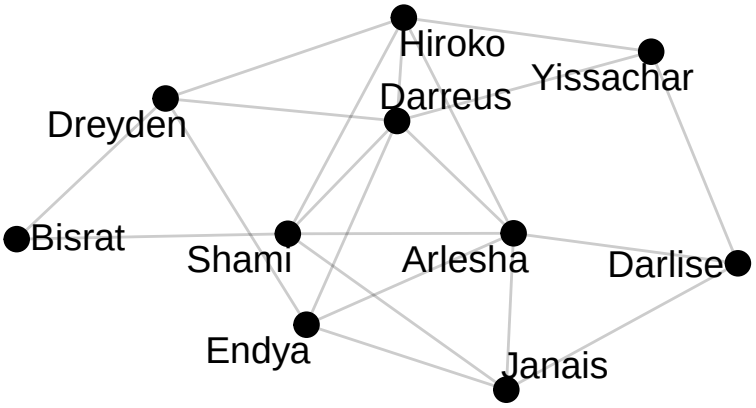
Key Terminology

There are four main terms that we need to define to sharpen up our understanding of a network:

1. **Node** (vertex): each of the units in the network
   - These are the 'points' in my very informal definition above
   - These are the entites that we want to model
     - For example, each node could be a person



2. **Edge** (tie): connection between nodes
   - These are the lines or arrows in my informal definition
   - Represent the connections between entities
     - For example could show who is friends with who



3. A **network** consists of a set of nodes and edges
   - i.e. a set of entities and their relationships
   - For example, the set of people *and* the friendship connections
4. **Attributes** give us more information about our network
   - *node attributes* are features of the each node
     - For example, they could be the name and sex of a person
     - *link attributes* are features of the connections between two nodes
     - For example, they could be strength of the connection, or the direction of the connection

Remark: A little more information about Edges

Edges represent the connections between nodes. We can extend our typology of edges with further definitions:

- Undirected Edges: the connection between two nodes is symmetric.
  - A is connected to B and B is connected to A
  - Example: Friends on Facebook
  - Generally, these type of edges are represented by lines
- Directed Edges: the connection between two nodes may be asymmetric
  - A is connected to B but B is not connected to A
  - Example: Jeff follows Susan on Twitter, but Susan does not follow Jeff
  - Generally, these type of edges are represented by arrows.
- Weighted: some edges have more strength than others
  - For example, Jeff and Susan have a stronger connection than Jeff and Fred because Jeff talks to Susan more regularly.
- Unweighted: the connections between all nodes have the same strength
  - It does not matter that Jeff talks to Susan more than he talks to Fred

In this class, unless we specify otherwise think of an edge as undirected and unweighted.

Representing a Network

With some important definitions in hand, we can now think about how we can represent network data. Essentially we need to keep track of who is in our network, and who is connected to whom. There are two popular ways to do this:

1. An Adjacency Matrix
2. Separate Tables of Nodes and Edges

Let's think about each in turn:

Adjacency Matrix

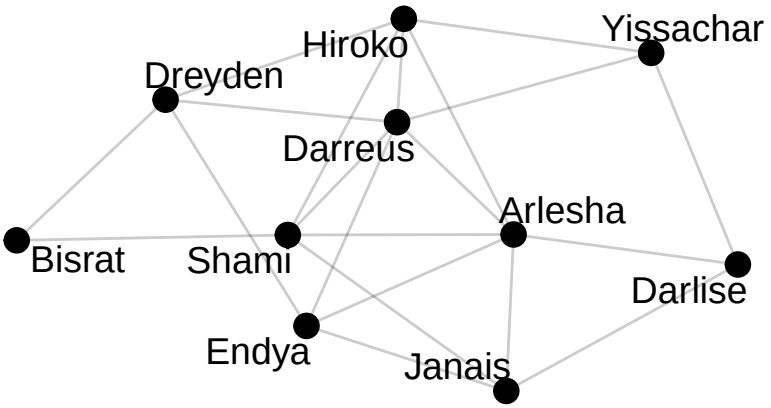
An adjacency matrix is a square $N \times N$ matrix, where N is the number of nodes. The rows and columns are indexed by the nodes of the network, and the entries of the matrix are either zero (if two nodes are not

connected) or one (if two nodes are connected).

OK, so that's *slightly* technical. Let's break it down. Think of a matrix as a table. We want the table to be square. Each person in our network is given a row, and a column. We keep the same name order for the rows as we do for the columns. The table is filled in based on whether a link exists between two nodes:

- If two people are connected, we put a "1" in the entry in the cell where the row and column intersect.
- If two people are not connected, we put a zero there.
  - We'll also put a zero where the row and column names are the same - that way a person is not connected to themselves.

Let's think about how we would create this table. Take the example network we saw above:



Let's start with Darlise. Darlise is not connected to herself, so we put a zero in the first row, first column. She is also not connected to Hiroko, Darreus, or Shami. So we put zeros in those entries along the first row. Darlise is connected to Janais, so we put a one there. And so on...

Continuing this process for all people in our network, our adjacency matrix becomes:

```
##           Darlise Hiroko Darreus Shami Janais Dreyden Bisrat Endya Arlesha
## Darlise         0      0       0     0      1       0      0     0       1
## Hiroko          0      0       1     1      0       1      0     0       1
## Darreus         0      0       0     1      0       1      0     1       1
## Shami           0      0       0     0      1       0      1     0       1
## Janais          0      0       0     0      0       0      0     1       1
## Dreyden         0      0       0     0      0       0      1     1       0
## Bisrat          0      0       0     0      0       0      0     0       0
## Endya           0      0       0     0      0       0      0     0       1
## Arlesha         0      0       0     0      0       0      0     0       0
## Yissachar       0      0       0     0      0       0      0     0       0
##           Yissachar
## Darlise           1
## Hiroko            1
## Darreus           1
## Shami             0
## Janais            0
## Dreyden           0
## Bisrat            0
## Endya             0
## Arlesha           0
## Yissachar         0
```

Representing a network as an adjacency matrix is a popular choice becuase it is a succinct way to analyse a large network and identify some key properties (we'll cover these properties in another week).

In most of what we will work on this semster, we'll stay away from this representation because it can be daunting if one hasn't looked at matrices for a while. It's good to see though, as if you read around this might be what an author presents you with.

Nodes and Edges Tables

The second way to represent a network is in two tables- one for nodes, and one for edges.

In the *nodes* table:

- Each row of the table is a unique node in the network
- Columns describe attributes of the nodes

And in the edges table (commonly called an edgelist):

- Each row is a connection between two nodes
- The first column represents where a connection starts
- The second is where the connection ends
- Subsequent columns may add edge attributes.

Let's again take the friendship network we have been looking at above and look at how we would construct the nodes table. It would look like this:

```
nodes %>%
    select(-age) %>%
    head(10)
```

```
## # A tibble: 10 x 2
##    name      sex
##    <chr>     <chr>
##  1 Darlise   F
##  2 Hiroko    F
##  3 Darreus   M
##  4 Shami     F
##  5 Janais    F
##  6 Dreyden   M
##  7 Bisrat    M
##  8 Endya     F
##  9 Arlesha   F
## 10 Yissachar M
```

We see each row is a node in our network. The variable `name` serves as a unique identifer of each node for us. There is an additional attribute, `sex` in the table telling us if the node is a male or female.

Turning to the edgelist table:

```
edgelist %>%
    head(21)
```

```
## # A tibble: 21 x 3
##    friend_1  friend_2 friendship_strength
##    <chr>     <chr>    <chr>
##  1 Shami     Arlesha  weak
##  2 Hiroko    Darreus  strong
##  3 Shami     Bisrat   weak
##  4 Arlesha   Hiroko   strong
##  5 Shami     Darreus  strong
##  6 Janais    Darlise  strong
##  7 Yissachar Darreus  strong
```

```
##  8 Dreyden   Hiroko   strong
##  9 Yissachar Darlise  strong
## 10 Endya     Arlesha  strong
## # ... with 11 more rows
```

We see that the first two columns are the names of the nodes where a connection runs between. Each row is a separate connection in the network. The third column is an edge attribute, telling us how strong the friendship is.

We will adopt this nodes and edgelist representation as the main way we represent a network.

Example Social Media Networks

We've now covered the basic terminology of networks. Networks abound us in the social media sphere (and online more broadly!) Here are some example of networks:

- Internet: websites / hyperlinks
- Twitter: users / retweets
- Twitter: users / following connections
- Twitter: hashtags / co-appeareance
- Facebook: friends / friendship connections
- Reddit: subreddits / users in common

A question for you to work through: Are the edges in these networks directed? Could they be weighted (if so, how?)

Packages for Network Analysis with R

It's time to change gears and start working with some network data in `R`. There is currently a wide range of packages that one can install and use in `R` for network analysis. We are going to focus on two:

- tidygraph
- ggraph

We will use these two packages because they have a similar philosophy underpinning their syntax to other packages you've worked with as part of the course preparation. Essentially they play along nicely with packages in the `tidyverse` - which means that we can readily integrate them with what we know from `dplyr` and `tidyr` when we work with the network data, and `ggplot` when we want to visualize the network.

Loading Example Network Data

In the `data` folder there are two csv files, `nodes.csv` and `edgelist.csv`. Combined, these two datasets comprise a network.

Let's load these csv files. We'll use `read_csv()` from `readr` to load the data:

```
library(readr)

nodes <- read_csv('data/nodes.csv')
edgelist <- read_csv('data/edgelist.csv')
```

If we look at the nodes data:

```
library(tibble) # to use 'glimpse'

glimpse(nodes)
```

```
## Rows: 10
## Columns: 3
```

```
## $ name <chr> "Darlise", "Hiroko", "Darreus", "Shami", "Janais", "Dreyden", ...
## $ age  <chr> "old", "old", "middle aged", "old", "middle aged", "middle age...
## $ sex  <chr> "F", "F", "M", "F", "F", "M", "M", "F", "F", "M"
```

We see that there are 10 nodes in the network. Each node is an individual, who has a `name`. Nodes also have attributes `age` which classifies the person as young, middle aged, and old, and `sex` which tells us there gender.

Turning to the edgelist data:

```
glimpse(edgelist)
```

```
## Rows: 21
## Columns: 3
## $ friend_1            <chr> "Shami", "Hiroko", "Shami", "Arlesha", "Shami",...
## $ friend_2            <chr> "Arlesha", "Darreus", "Bisrat", "Hiroko", "Darr...
## $ friendship_strength <chr> "weak", "strong", "weak", "strong", "strong", "...
```

We can see that there are 21 connections between the 10 nodes. Each edge also has a `friendship_strength` which takes the value "strong" or "weak" indicating the how strong a friendship is.

Remark: You may have noticed that the data you have now loaded **is** the that we used to describe some of the basic network concepts earlier.

Creating a Network Graph

The next step we want to take is to tell `R` to treat the nodes and edges as a `network`. Unless we explictly make this connection, `R` will treat these two datasets as exactly that - two separate datasets that are not related to each other.

As mentioned above, we will do all of our network analysis using the `tidygraph` package.

First, we load the tidygraph package:

```
library(tidygraph)
```

Next we use the `tbl_graph` function from the `tidygraph` package to transform our list of nodes and edges into a network:

```
grph <- tbl_graph(
    nodes = nodes,
    edges = edgelist,
    directed = FALSE
)
```

We use `directed = FALSE` because the edgelist does not specify that the edges are directed.

To get a sense of what has just happened, let's take a look at the object we just created by printing it's contents to screen:

```
grph
```

```
## # A tbl_graph: 10 nodes and 21 edges
## #
## # An undirected simple graph with 1 component
## #
## # Node Data: 10 x 3 (active)
##   name    age         sex
##   <chr>   <chr>       <chr>
## 1 Darlise old         F
## 2 Hiroko  old         F
```

```
## 3 Darreus middle aged M
## 4 Shami   old         F
## 5 Janais  middle aged F
## 6 Dreyden middle aged M
## # ... with 4 more rows
## #
## # Edge Data: 21 x 3
##    from    to friendship_strength
##   <int> <int> <chr>
## 1     4     9 weak
## 2     2     3 strong
## 3     4     7 weak
## # ... with 18 more rows
```

Notice that when we use `tbl_graph()` it takes the edgelist and turns the names of the nodes an edge is connecting into numeric data and renames the edges being connected to `from` and `to`. The numeric values in `from` and `to` are the row numbers in the node data.

Remark: If one only has access to an edgelist (i.e not a list of nodes and attributes) we can still work with the `tidygraph` package:

```
grph2 <- tbl_graph(
    edges = edgelist,
    directed = FALSE
)
```

Then if we look at the object created:

```
grph2
```

```
## # A tbl_graph: 10 nodes and 21 edges
## #
## # An undirected simple graph with 1 component
## #
## # Node Data: 10 x 1 (active)
##   name
##   <chr>
## 1 Shami
## 2 Arlesha
## 3 Hiroko
## 4 Darreus
## 5 Bisrat
## 6 Janais
## # ... with 4 more rows
## #
## # Edge Data: 21 x 3
##    from    to friendship_strength
##   <int> <int> <chr>
## 1     1     2 weak
## 2     3     4 strong
## 3     1     5 weak
## # ... with 18 more rows
```

We see that `R` creates a node list from the edgelist by finding all unique nodes.

Plotting the Network: First Steps

To plot network data we are going to use the `gggraph` package:

```
library(ggraph)
```

The advantage of `ggraph` is that it builds upon `ggplot2` in terms of syntax. This is great when we don't want to learn a new way of thinking about building a graph. We can adopt a similar mentality from `ggplot` and 'learn' a handful of new commands that are specific to network plots.

To start constructing a plot, we pass a network into a plotting call. We named the network `grph`, and the `ggraph()` is the function that will set up our network plot:

```
grph %>%
    ggraph()
```

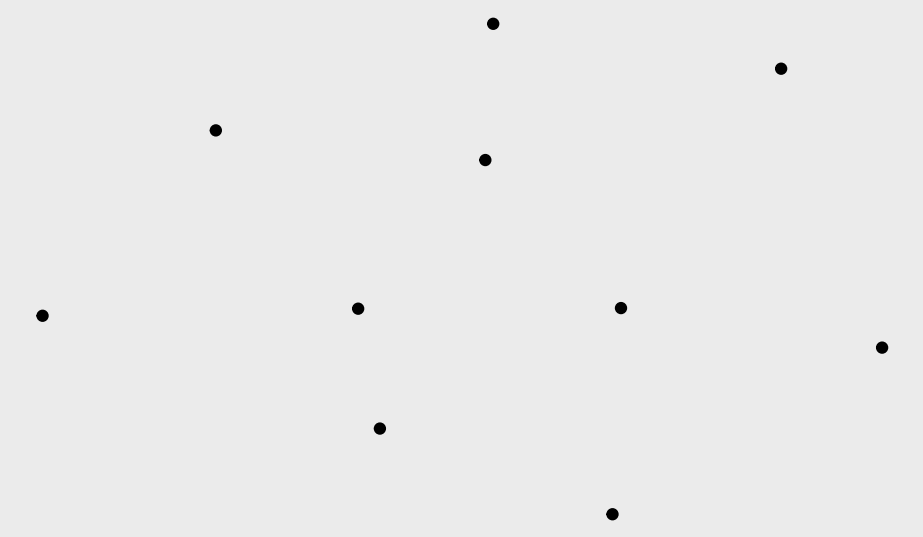
```
## Using 'stress' as default layout
```


By default nothing is plotted. This is the same as what happens with `ggplot` - we must build up the plot ourselves by telling `R` what to plot.

First, we can add nodes:

```
grph %>%
    ggraph() +
    geom_node_point()
```

```
## Using 'stress' as default layout
```

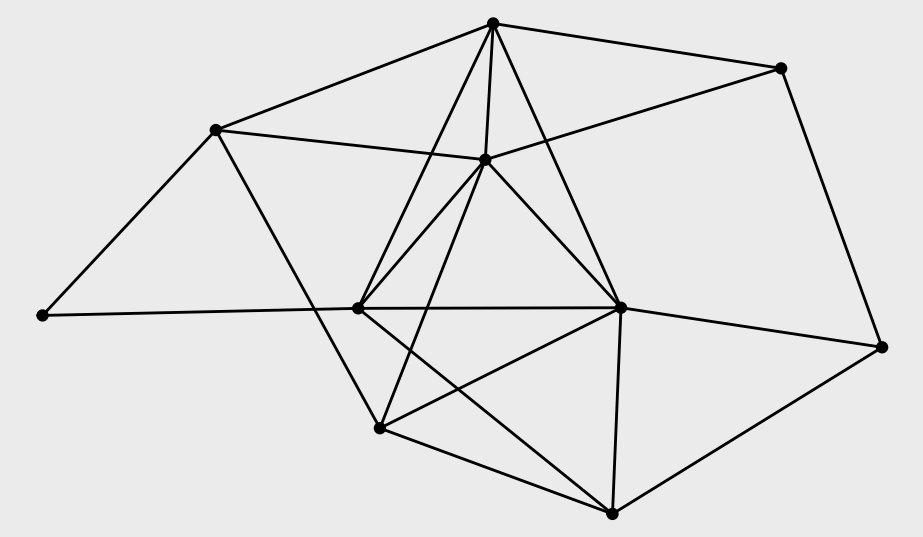


Great, so now the nodes are placed onto the plot. We see that they are placed on the plot in some kind of "scattered" fashion.

Next, we can add the edges:

```
grph %>%
    ggraph() +
    geom_node_point() +
    geom_edge_link()
```

```
## Using 'stress' as default layout
```

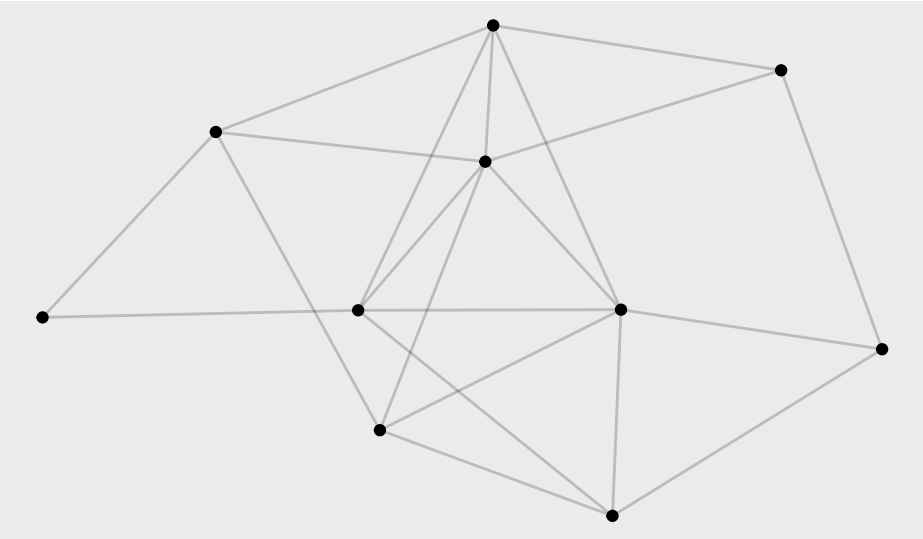


This is progress! Now we have the nodes and edges plotted, and the graph starts to look like a "typical" network graph.

Looking at the graph above, the nodes and edges are both the same color, in our case black. This makes the nodes a little hard to visualize since. We can soften the "blackness" of the edges by de-intensifying the opacity of them. This can be done by changing the `alpha` value of the edges. An alpha value of 1 means totally opaque, and an alpha value of 0 means totally transparent. Let's set `alpha=0.2` which makes the edges a more grey-ish color

```
grph %>%
    ggraph() +
    geom_node_point() +
    geom_edge_link(alpha = 0.2)
```

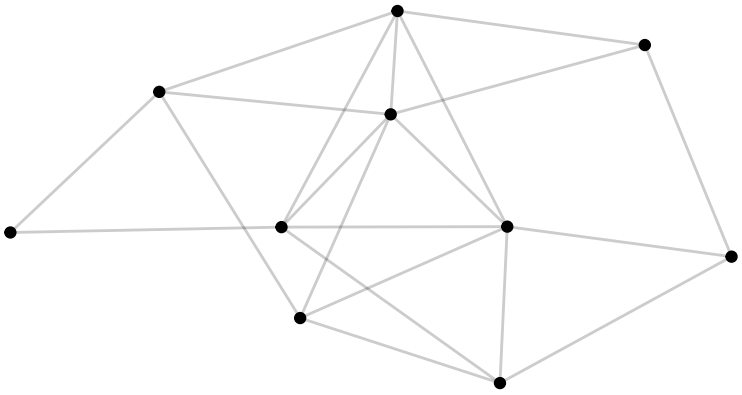
```
## Using 'stress' as default layout
```



The next step we can take is removing the grey background. We use `theme_graph()` - to get a white background.

```
grph %>%
    ggraph() +
    geom_node_point() +
    geom_edge_link(alpha = 0.2) +
    theme_graph()
```

```
## Using 'stress' as default layout
```



Customizing the Nodes and Links Shape and Color

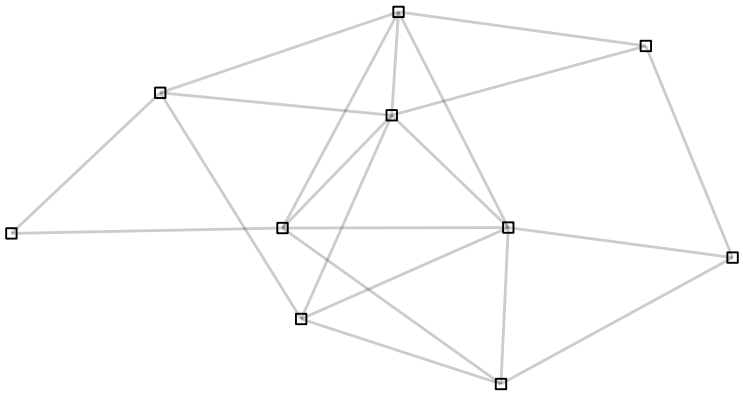
In the previous section we built a rather basic plot that visualized the friendship network. We will now explore how we can extend this graph by customizing parts of the plot.

Changing the node shape and size

We can change the shape of the nodes away from being circular. There's many choices, here's one example:

```
grph %>%
    ggraph() +
    geom_node_point(shape = 0) +
    geom_edge_link(alpha = 0.2) +
    theme_graph()
```

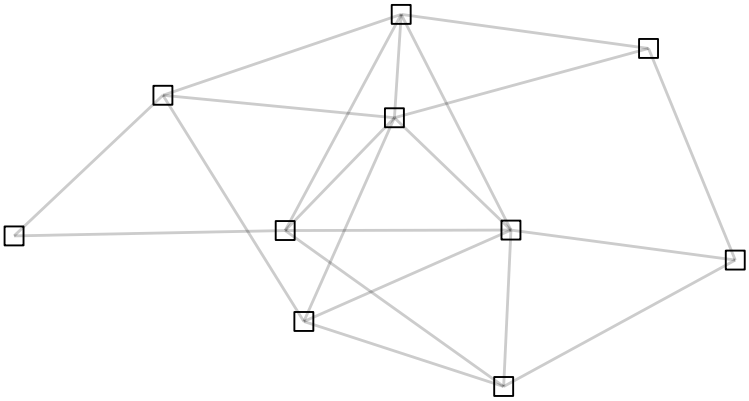
```
## Using 'stress' as default layout
```



And we can change the size of all nodes as follows:

```
grph %>%
    ggraph() +
    geom_node_point(shape = 0, size = 3) +
    geom_edge_link(alpha = 0.2) +
    theme_graph()
```

```
## Using 'stress' as default layout
```

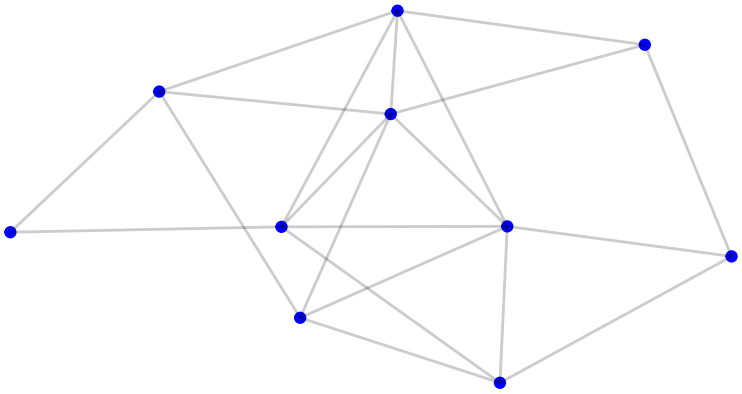


Changing node color

We can set the color of all nodes as follows:

```
grph %>%
    ggraph() +
    geom_node_point(color = "blue") +
    geom_edge_link(alpha = 0.2) +
    theme_graph()
```

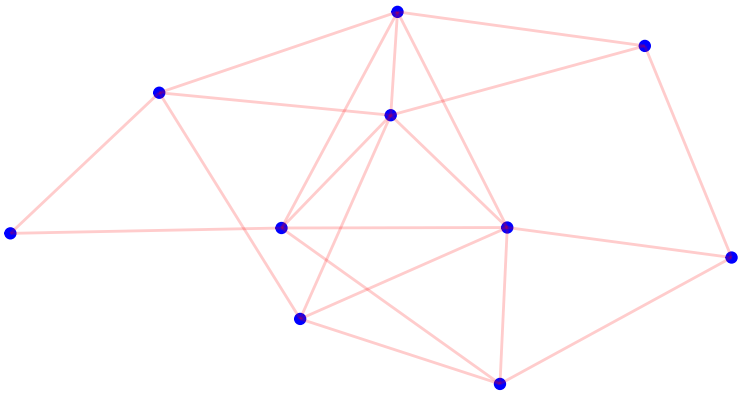
```
## Using 'stress' as default layout
```



We could also change the color of the edges:

```
grph %>%
    ggraph() +
    geom_node_point(color = "blue") +
    geom_edge_link(alpha = 0.2, color = "red") +
    theme_graph()
```

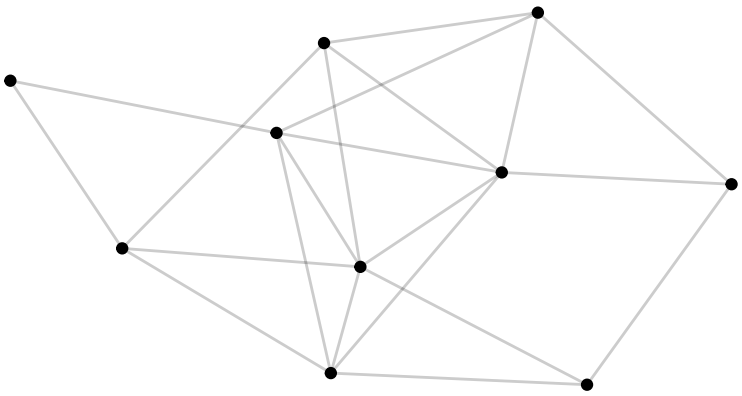
```
## Using 'stress' as default layout
```



Changing the Layout

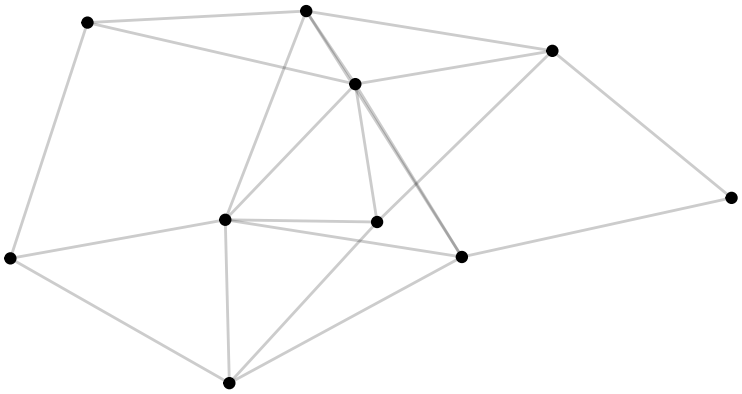
So far, we have changed the color of nodes and points, but left the layout - in terms of where the nodes are located on the plot fixed. We can change this layout too. By default, `ggraph` uses what's known as the "stress" layout (think of this as an algorithm that decides where to locate nodes). The "stress" layout generally produces a pleasant result, but sometimes we will want to try out some others and see what the result becomes. One alternative is a layout algorithm called "kk". We can choose the "kk" layout as follows:

```
grph %>%
    ggraph(layout = "kk") +
    geom_node_point() +
    geom_edge_link(alpha = 0.2) +
    theme_graph()
```



OK, so in this case one could argue that not much happened. Let's try another, this time "fr":

```
grph %>%
    ggraph(layout = "fr") +
    geom_node_point() +
    geom_edge_link(alpha = 0.2) +
    theme_graph()
```



It *is* different, but not neccessarily better. Essentially one needs to play around a little and find the best layout for the network at hand. The question then is: what layouts can one choose from? Here's a non-exhaustive list:

"dh", "drl", "fr", "gem", "graphopt", "kk", "lgl", "mds", "sugiyama", "bipartite", "star", "stress" or "tree"

Try them out for yourself.

Adding Attributes to the Plot

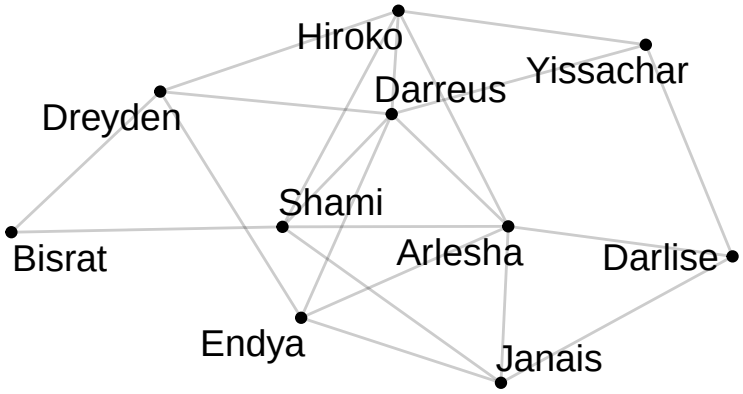
Once we have got a basic plot constructed, we can start to add some further detail utilizing the attributes of the nodes and edges. Let's explore these now.

Node attributes

We can add the names of the nodes as follows:

```
grph %>%
    ggraph() +
    geom_node_point() +
    geom_node_text(aes(label = name), size = 5, repel = TRUE) +
    geom_edge_link(alpha = 0.2) +
    theme_graph()
```

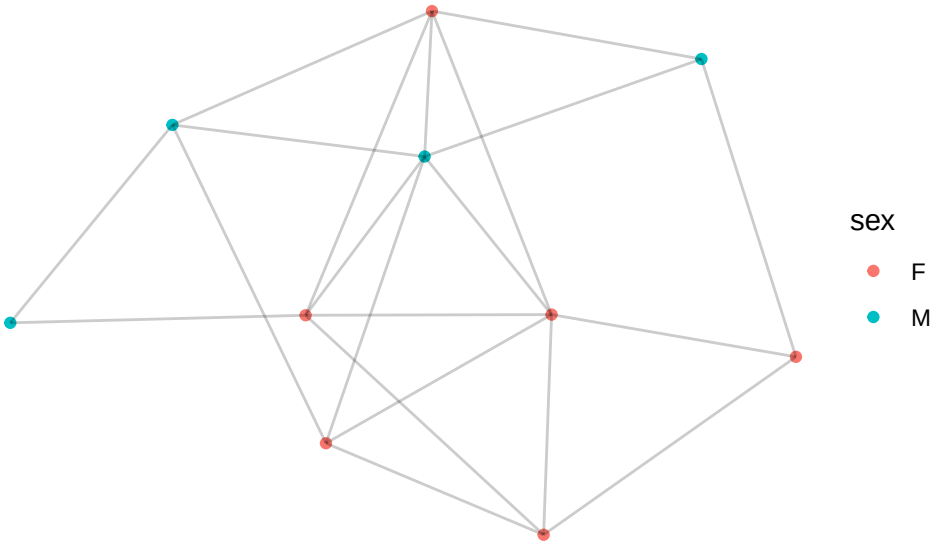
```
## Using 'stress' as default layout
```



Instead of adding the names to the nodes, we could color the nodes according to a node attribute. For example we could have different color nodes for each sex. We do that by constructing an aesthetic mapping:

```
grph %>%
    ggraph() +
    geom_node_point(aes(color = sex)) +
    geom_edge_link(alpha = 0.2) +
    theme_void()
```

```
## Using 'stress' as default layout
```

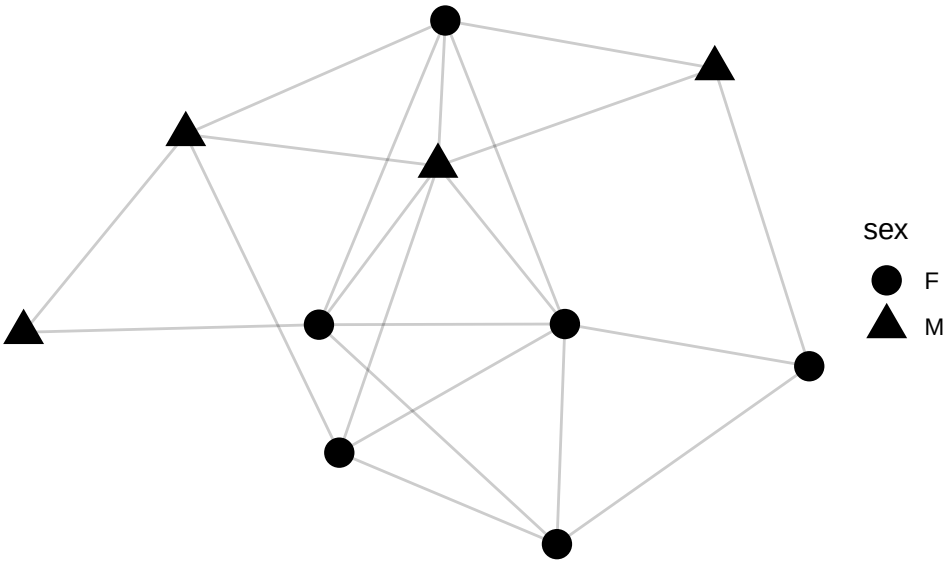


These red and blue-ish colors are the `ggplot` defaults. You can change the color mappings the same way you would modify any `ggplot` color mapping (see here for some examples)

In the same way we modified node color, we could modify node shape to depend on sex:

```
grph %>%
    ggraph() +
    geom_node_point(aes(shape = sex), size = 5) +
    geom_edge_link(alpha = 0.2) +
    theme_void()
```

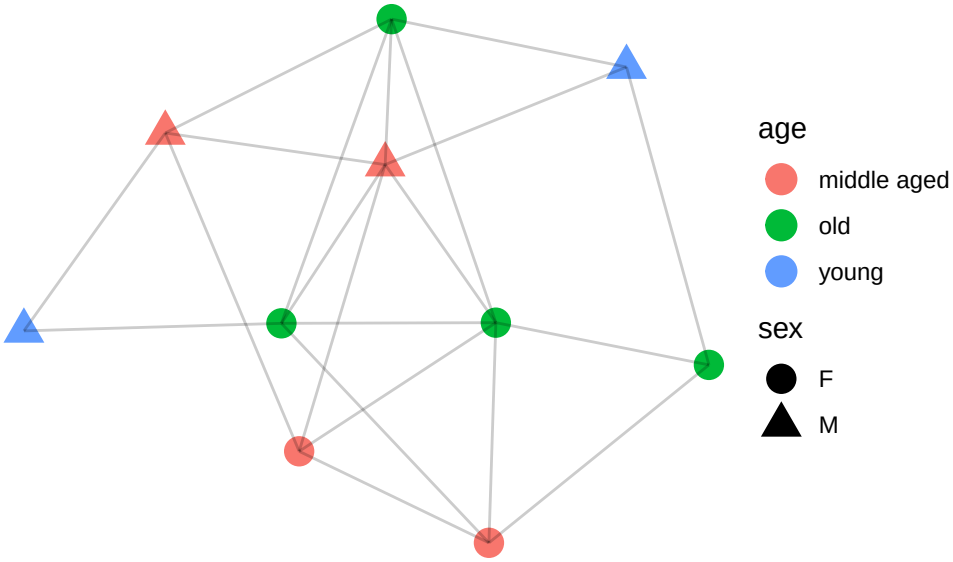
```
## Using 'stress' as default layout
```



And we could use multiple attributes at once:

```
grph %>%
    ggraph() +
    geom_node_point(aes(shape = sex, color = age), size = 5) +
    geom_edge_link(alpha = 0.2) +
    theme_void()
```

```
## Using 'stress' as default layout
```



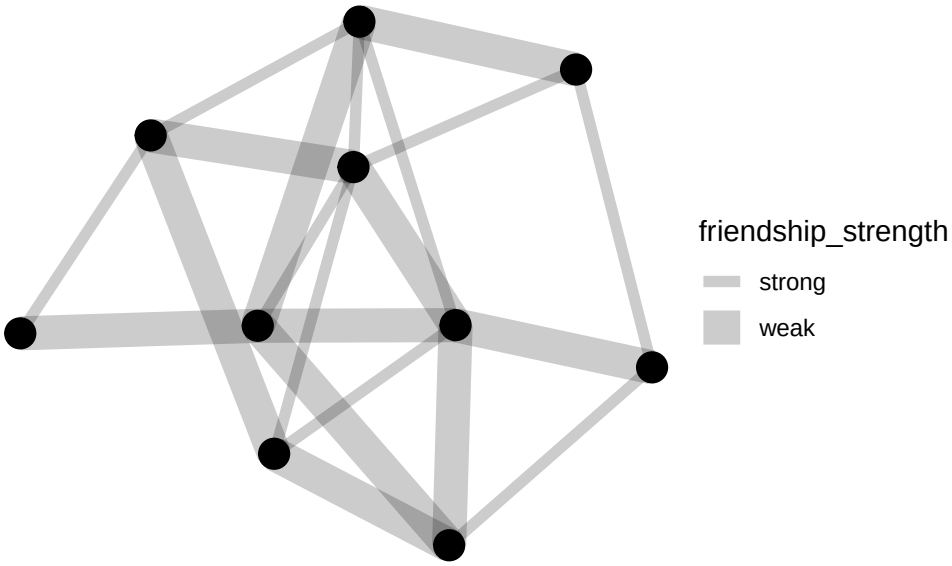
Edge Attributes

We can also modify the edges. Recall that in the edgelist there was the variable `friendship_strength`. Like with nodes, one can set the color based on an attribute. Perhaps more interestingly, we could change the thickness:

```
grph %>%
    ggraph() +
    geom_node_point(size = 5) +
```

```
    geom_edge_link(aes(width = friendship_strength), alpha = 0.2) +
    theme_void()
```

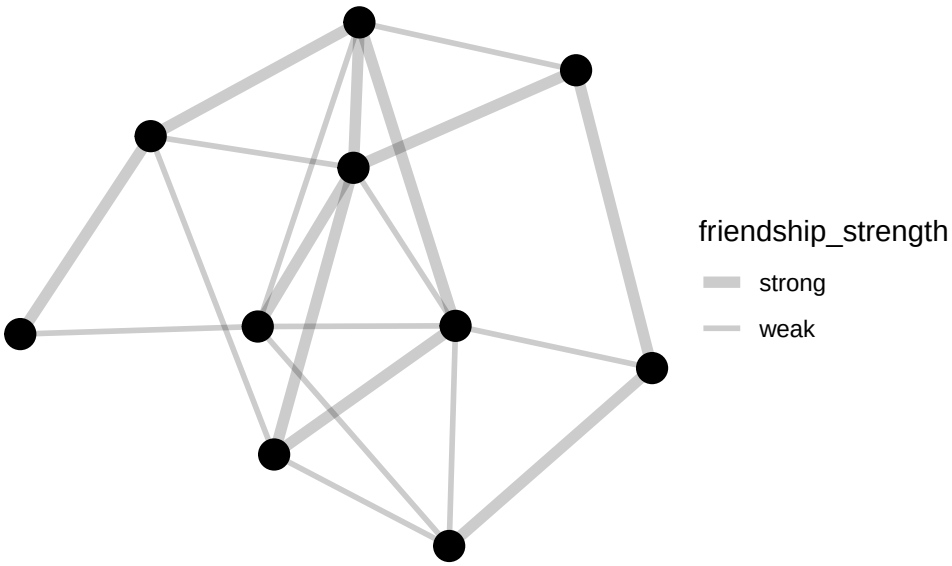
```
## Using 'stress' as default layout
```



This worked, but it might make sense to have *strong* ties have a thicker edge than weak ones. To do this we have to manually set the widths:

```
grph %>%
    ggraph() +
    geom_node_point(size = 5) +
    geom_edge_link(aes(width = friendship_strength), alpha = 0.2) +
    scale_edge_width_discrete(range = c(2, 1)) +
    theme_void()
```

```
## Using 'stress' as default layout
```



Creating New Attributes?

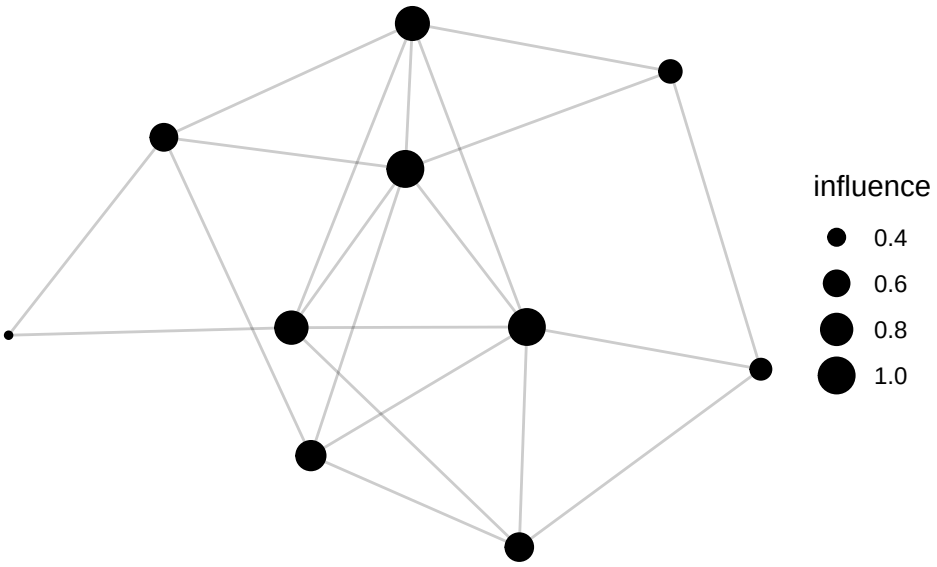
The modifications we have made so far have been using attributes of nodes and edges that we have inherited in the dataset we loaded.

For example, we could size the nodes based on how influential a node could be. Think of influence loosely as how many edges a node has (we'll be more precise on this in a future class). One way to measure influence is computing something called `centrality_authority`.

An advantage of `tidygraph` is that we can pass the graph object we created `grph` into a sequence of `dplyr` verbs. In this case, I will take `grph` and create a attribute called `influence`. Once this new variable is created, I can scale the node sizes based on this new measure:

```
grph %>%
    # create the new variable
    mutate(influence = centrality_authority()) %>%
    # pass this data into a plotting call
    ggraph() +
    geom_node_point(aes(size = influence)) +
    geom_edge_link(alpha = 0.2) +
    scale_edge_width_discrete(range = c(2, 1)) +
    theme_void()
```

```
## Using 'stress' as default layout
```

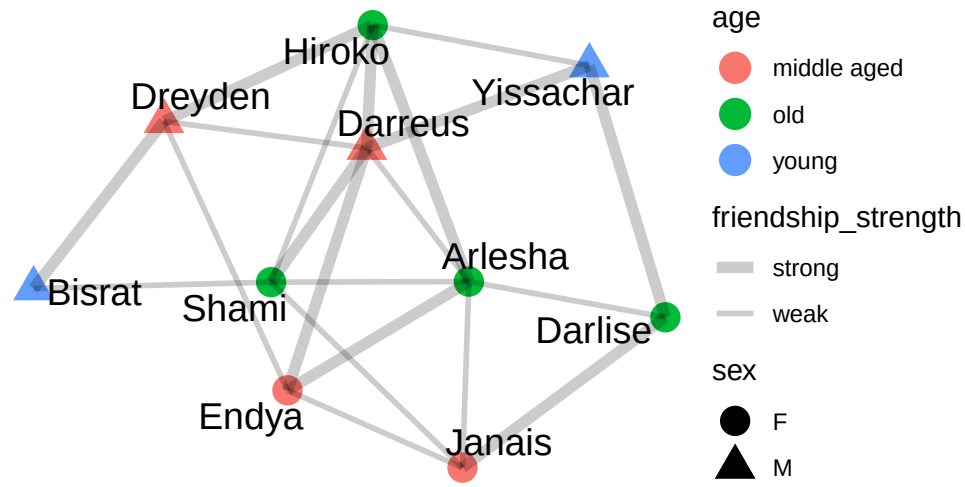


A 'Final Plot'

Let's take what we have learned, and construct a final plot that utilizes some of the different choices we have looked at in this document. We also add a title and a subtitle to the graph for completeness:

```
grph %>%
    ggraph() +
    geom_node_point(aes(shape = sex, color = age), size = 5) +
        geom_node_text(aes(label = name), size = 5, repel = TRUE) +
        geom_edge_link(aes(width = friendship_strength), alpha = 0.2) +
        scale_edge_width_discrete(range = c(2, 1)) +
    theme_void() +
    ggtitle('Friendship Connections', subtitle = "Simulated Network")
```

```
## Using 'stress' as default layout
```



Whether that is a "pretty" graph or not, I will leave you to judge as the reader. At a minimum, I hope it shows how much control one has using `ggraph` - but it is up to the graph maker to use that control wisely!

Summary

In this document we:

- Introduced basic terminology to discuss a network
- Took a pair of datasets (nodes and egdes) and made a network out of them
- Plotted the network

There's much more we can do with network data, but what we have covered here is the main building blocks. Your next step, will be to pair what you have learned about networks with your knowledge about accessing Twitter data (covered in here) to construct networks from tweets. One can build some very interesting visualizations of Twitter networks with the tools we have covered. I hope you are looking forward to seeing what is possible!

Later in the course, we will look at ways to describe features of the network, and do some further analysis about detecting subcommunities, and identifying influential nodes.

Bonus Material: Creating a Simulated Network

The network data we have been using so far is a simulated network. This means that I have 'created' the nodes and edges myself. In this section, I demonstrate how I created the simulated network in `R`.

Remark: This is not the most efficient way to simulate some network data. I've adopted this method since it utilizes commands you should be familiar with from the course preparation (or you could easily search through the help menus or google to find).

First, I needed to create a set of nodes. The network I wanted to construct has 10 nodes, so I created a variable that stores how large I wanted my network to be:

```
n_nodes <- 10
```

The nodes were named. To create names, I drew a sample of names from a list of baby names that were popular in the USA between 1980 and 2017. These names are stored in the R package `babynames`:

```
library(babynames)
```

The `babynames` package has a dataset called `babynames`, let's look at the first few rows:

```
head(babynames)
```

```
## # A tibble: 6 x 5
##    year sex   name          n   prop
##   <dbl> <chr> <chr>     <int>  <dbl>
## 1  1880 F     Mary       7065 0.0724
## 2  1880 F     Anna       2604 0.0267
## 3  1880 F     Emma       2003 0.0205
## 4  1880 F     Elizabeth  1939 0.0199
## 5  1880 F     Minnie     1746 0.0179
## 6  1880 F     Margaret   1578 0.0162
```

We can see there are a list of names, the sex of the name, and some variables that tell us about how popular a name is in each year.

Now I want to create some nodes. I do this by sampling `n_nodes` datapoints from the data. When I do this, I first restrict the data to only contain unique names so I don't have a bunch of nodes with the same name. Once I have the nodes (which come with a gender attached - since it is a column in the data), I create a second attribute `age`. Age is a categorical variable that is based on the last time a name was a popular babyname. I treat this as the 'age' of a person in a friendship network.

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
set.seed(1234567890)

nodes <- babynames %>%
         distinct(name, .keep_all = TRUE) %>%
         sample_n(n_nodes) %>%
         mutate(age = case_when(
             year >= 2000 ~ "young",
             year < 2000 & year >= 1980 ~ "middle aged",
             TRUE ~ "old"
         )
         ) %>%
         select(name, age, sex)
```

Now that there is nodes, we need to create some connections between nodes, i.e. the edges.

I simulated the links as follows:

- I set the max number of edges I wanted in my data, `n_links_max`
- I created two data sets `from` and `to` by drawing from the `nodes` with replacement. Each of these datasets got a numeric id - `link_id`, which was a potential edge in the network.

- I joined the data, using the `link_id`. This gives me potential edges since theres a starting node and an ending node.
- I filtered out links that were:
  - from and to the same person (so people aren't connected to themselves)
  - duplicates of existing links
  - links that were the reverse of each other (A to B and B to A)

The code below does these steps:

```
library(tibble) # need this to make a row id

n_links_max <- 30

from <- nodes %>%
    select(name) %>%
    sample_n(n_links_max, replace = TRUE) %>%
    rowid_to_column("link_id") %>%
    rename(friend_1 = name)

to <- nodes %>%
    select(name) %>%
    sample_n(n_links_max, replace = TRUE) %>%
    rowid_to_column("link_id") %>%
    rename(friend_2 = name)

edgelist <- from %>%
         inner_join(to, by = c("link_id")) %>%
         filter(friend_1 != friend_2) %>%
         distinct(friend_1, friend_2) %>%
        # if the same pair is included, but the other way around, remove them
         filter(!duplicated(paste0(pmax(friend_1, friend_2), pmin(friend_1, friend_2))))
```

Finally, I added an attribute to each edge. Each edge was assigned a "friendship_strength", which could take the values "strong", or "weak". Weak connections occur approx 70 percent of the time, strong approx 30 percent.

```
friendship_types <- c("strong", "weak")
prop_types <- c(0.3, 0.7)

edgelist <- edgelist %>%
    mutate(friendship_strength = sample(friendship_types,
                                        n(),
                                        replace = TRUE,
                                        prob = prop_types)
           )
```

I then saved this data, so that I could use it later:

```
library(readr)

write_csv(nodes, "data/nodes.csv")
write_csv(edgelist, "data/edgelist.csv")
```

Acknowledgements

This set of notes borrows quite heavily from the work of others. Material has been borrowed and adapted from the following sources:

- "How to model a social network with R" by Ayman Bari
- "Discovery in Large-Scale Social Media Data" by Pablo Barbera