Lab 8 - Networks

Sebastian Montesinos

For class Tuesday, April 5

Lab Purpose

This lab is designed to help you develop skills involving the analysis of network data (graphs). We'll explore a dataset on migration between countries from 1960 to 2000 and also a dataset based on character interactions in George R.R. Martin's `A Storm of Swords`.

The lab focuses on two main packages:

- igraph - this package has a lot of functionality for analysis of networks, including clustering algorithms - however, it doesn't produce the best visuals
- ggnetwork - this package helps with visualizations of networks (convert igraph objects so they can be plotted with ggplot2) and provides other useful functionality (network geometrics such as `geom_edges` and `geom_nodes`)

As usual, make sure you load each package in the `setup` code chunk above, after installing once (if necessary). You should have `igraph` installed from the prep already.

1 - Country Migration Network

Data and Setup

The following dataset contains migration counts for decades between 1960 and 2000 between the origin (`origincode`) and destination (`destcode`) countries given in the data. The lab is set up to look at the migration flows of females in 2000, but you can change this to males and/or any year you wish in the appropriate wrangling chunk below.

```r
# Read in dataset from data subfolder
migration_flows <- read_csv("data/migration-flows.csv")

# What are the variables?
glimpse(migration_flows)
```

```
Rows: 107,184
Columns: 8
$ sex        <chr> "Male", "Male", "Male", "Male", "Male", "Male", "Male", "Ma~
$ destcode   <chr> "FRA", "FRA", "FRA", "FRA", "FRA", "FRA", "FRA", "FRA", "FR~
$ origincode <chr> "AFG", "DZA", "AUS", "AUT", "AZE", "BLR", "BLZ", "BEN", "AL~
$ Y2000      <dbl> 923, 425229, 9168, 7764, 118, 245, 391, 166, 10017, 0, 122,~
```

```
$ Y1990      <dbl> 91, 861691, 903, 2761, 12, 88, 38, 397, 3586, 0, 43, 10907,~
$ Y1980      <dbl> 55, 794288, 1483, 4686, 20, 26, 25, 4409, 4, 0, 436, 76, 0,~
$ Y1970      <dbl> 29, 723746, 1906, 4861, 4, 0, 22, 5736, 17, 0, 0, 130, 0, 0~
$ Y1960      <dbl> 1471, 521679, 14614, 12375, 188, 390, 623, 233, 15967, 0, 1~
```

```r
# View a few rows to get a sense of the data
head(migration_flows, n = 10)
```

```
# A tibble: 10 x 8
   sex   destcode origincode  Y2000  Y1990  Y1980  Y1970  Y1960
   <chr> <chr>    <chr>       <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
 1 Male  FRA      AFG           923     91     55     29   1471
 2 Male  FRA      DZA        425229 861691 794288 723746 521679
 3 Male  FRA      AUS          9168    903   1483   1906  14614
 4 Male  FRA      AUT          7764   2761   4686   4861  12375
 5 Male  FRA      AZE           118     12     20      4    188
 6 Male  FRA      BLR           245     88     26      0    390
 7 Male  FRA      BLZ           391     38     25     22    623
 8 Male  FRA      BEN           166    397   4409   5736    233
 9 Male  FRA      ALB         10017   3586      4     17  15967
10 Male  FRA      ASM             0      0      0      0      0
```

```r
tail(migration_flows, n = 10)
```

```
# A tibble: 10 x 8
   sex    destcode origincode Y2000 Y1990 Y1980 Y1970 Y1960
   <chr>  <chr>    <chr>      <dbl> <dbl> <dbl> <dbl> <dbl>
 1 Female ZWE      VUT            0     0     0     0     0
 2 Female ZWE      VEN            0     0     0     0     0
 3 Female ZWE      VNM            5    10    10     9     9
 4 Female ZWE      VIR            0     0     0     0     0
 5 Female ZWE      VGB            0     0     0     0     0
 6 Female ZWE      WLF            0     0     0     0     0
 7 Female ZWE      PSE            0     1     0     0     0
 8 Female ZWE      YEM            0     0     0     0     0
 9 Female ZWE      ZMB        10451 21561 20336 19180 17640
10 Female ZWE      ZWE            0     0     0     0     0
```

First, we need to do some very minor wrangling to get our data ready for analyzing as a network: (1) include only rows with *positive* counts of female migration in 2000 and (2) keep only the variables `destcode`, `origincode`, and `Y2000`. How many rows are in this dataset?

```r
migration_flows_choice <- migration_flows %>%
 filter(sex == "Female", Y2000 > 0) %>%
 select(origincode, destcode, Y2000)
```

This dataframe can be used to create a directional network object (called an "igraph") with edges indicating migration from the origin county to a destination country for the migration network of females in 2000.

We'll be using `graph_from_data_frame()` from the **igraph** package. The order of the columns matters for directed graphs (first is the origin; second is the destination; third, if any, is an edge attribute).

```r
migration_igraph <- graph_from_data_frame(migration_flows_choice,
           directed = TRUE)
summary(migration_igraph)
```

```
IGRAPH ccb31d7 DN-- 226 13805 --
+ attr: name (v/c), Y2000 (e/n)
```

Then we can get basic statistics about the network.

```r
# Get descriptions and counts of vertices
V(migration_igraph) # not necessarily useful to print
```

```
+ 226/226 vertices, named, from ccb31d7:
  [1] AFG DZA AUS AUT AZE BLR BLZ BEN ALB AND AGO AIA ATG ARG ARM ABW BHS BHR
 [19] BGD BRB BEL BMU BTN BOL BIH BWA BRA BRN BGR BFA BDI KHM CMR CAN CPV CYM
 [37] CAF TCD CHL CHN COL COM COD COG CRI CIV HRV CUB CYP CZE DNK DJI DMA DOM
 [55] ECU EGY SLV GNQ ERI EST ETH FRO FLK FJI FIN GUF PYF GAB GMB GEO DEU GHA
 [73] GIB GRC GRL GRD GLP GTM GIN GNB GUY HTI HND HKG HUN ISL IND IDN IRN IRQ
 [91] IRL ISR ITA JAM JPN JOR KAZ KEN PRK KOR KWT KGZ LAO LVA LBN LSO LBR LBY
[109] LIE LTU LUX MAC MKD MDG MWI MYS MLI MLT MTQ MRT MUS MEX MDA MCO MNG MAR
[127] MOZ MMR NAM NPL NLD ANT NCL NZL NIC NER NGA NOR OMN PAK PAN PNG PRY PER
[145] PHL POL PRT PRI QAT REU ROM RUS RWA SPM WSM SMR STP SAU SEN SCG SYC SLE
[163] SGP SVK SVN SOM ZAF ESP LKA KNA LCA VCT SDN SUR SWZ SWE CHE SYR TWN TJK
+ ... omitted several vertices
```

```r
vcount(migration_igraph)
```

```
[1] 226
```

```r
# Get descriptions and counts of edges
E(migration_igraph) # not necessarily useful to print
```

```
+ 13805/13805 edges from ccb31d7 (vertex names):
 [1] AFG->FRA DZA->FRA AUS->FRA AUT->FRA AZE->FRA BLR->FRA BLZ->FRA BEN->FRA
 [9] ALB->FRA AND->FRA AGO->FRA AIA->FRA ATG->FRA ARG->FRA ARM->FRA ABW->FRA
[17] BHS->FRA BHR->FRA BGD->FRA BRB->FRA BEL->FRA BMU->FRA BTN->FRA BOL->FRA
[25] BIH->FRA BWA->FRA BRA->FRA BRN->FRA BGR->FRA BFA->FRA BDI->FRA KHM->FRA
[33] CMR->FRA CAN->FRA CPV->FRA CYM->FRA CAF->FRA TCD->FRA CHL->FRA CHN->FRA
[41] COL->FRA COM->FRA COD->FRA COG->FRA CRI->FRA CIV->FRA HRV->FRA CUB->FRA
[49] CYP->FRA CZE->FRA DNK->FRA DJI->FRA DMA->FRA DOM->FRA ECU->FRA EGY->FRA
[57] SLV->FRA GNQ->FRA ERI->FRA EST->FRA ETH->FRA FRO->FRA FLK->FRA FJI->FRA
[65] FIN->FRA GUF->FRA PYF->FRA GAB->FRA GMB->FRA GEO->FRA DEU->FRA GHA->FRA
[73] GIB->FRA GRC->FRA GRL->FRA GRD->FRA GLP->FRA GTM->FRA GIN->FRA GNB->FRA
+ ... omitted several edges
```

```r
ecount(migration_igraph)
```

```
[1] 13805
```

Get edge attribute, change to your year if different
edge_attr(migration_igraph, name = "Y2000") %>%
 head()
```

```
[1]    844 201387   8385   7100    108    224
```

part a - How many nodes are in this network? How many edges?

Solution: There are 226 nodes in this network, and 13805 edges.

```
# Alternative network construction

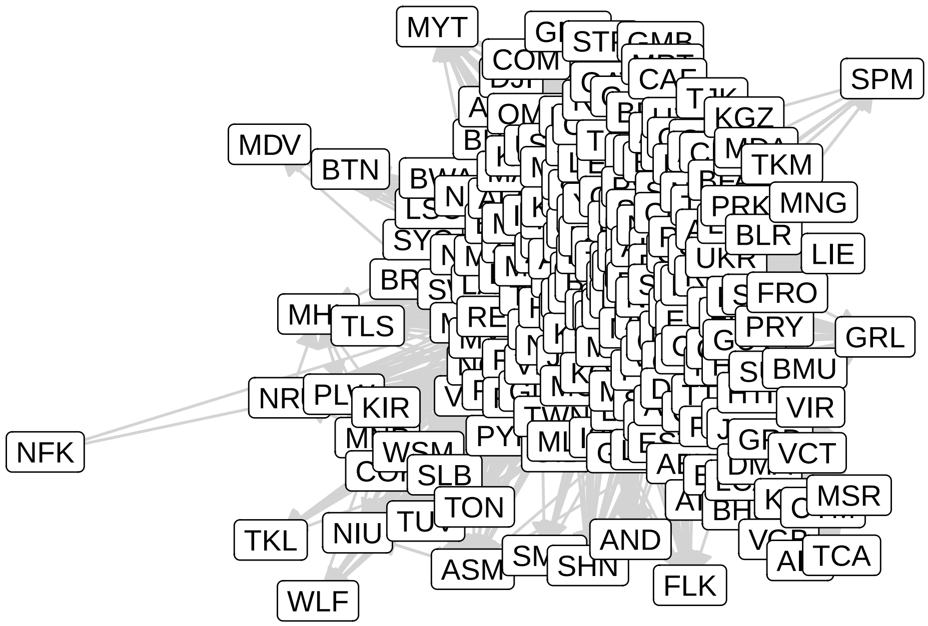
# Could also graph from edge list but need to
## 1. create edge list matrix
migration_el <- migration_flows_choice %>%
 select(origincode, destcode) %>%
 as.matrix()

## 2. create edge graph structure
migration_el_igraph <- graph_from_edgelist(el = migration_el, directed = TRUE)

## 3. add Y2000 as an edge attribute
migration_el_igraph <- set_edge_attr(graph = migration_el_igraph,
                   name = "Y2000",
                   value = migration_flows_choice$Y2000)

summary(migration_el_igraph)
```

We can plot the network with igraph, but the result isn't very visually appealing.

```
# The graph plotting actually needs a seed in igraph to be reproducible
set.seed(231)
plot(migration_igraph)
```

While this can work for small graphs, we can create a better visualization of this network using `ggnetwork()` to convert the igraph object to a network object, and `ggplot()` to plot the network graph.

```
migration_network <- ggnetwork(migration_igraph)
head(migration_network)
```

```
          x         y name      xend      yend Y2000
1 0.0000000 0.2592951  NFK 0.5830937 0.4386270    91
2 0.0000000 0.2592951  NFK 0.6341176 0.5456436     2
3 0.2679849 0.7949914  MDV 0.6239215 0.5849770     2
4 0.2679849 0.7949914  MDV 0.5287109 0.6684206     1
5 0.2679849 0.7949914  MDV 0.6245632 0.6583652     5
6 0.2679849 0.7949914  MDV 0.6577961 0.5185068    11
```

```
ggplot(data = migration_network, aes(x = x, y = y,
          xend = xend, yend = yend)) +
 geom_edges(arrow = arrow(type = "closed", length = unit(8, "pt")),
```
```

```
    color = "lightgray") +
 geom_nodes() +
 geom_nodelabel(aes(label = name)) +
 theme_blank()
```



There are still too many countries for this to be really useful (unless you want to make it interactive and zoom in). So let's examine a subset of countries. You can pick the countries you want to explore. Be sure you pick a subset of 10 countries.

The countries are all denoted by their 3 letter UN code, which you can explore here:

https://unstats.un.org/unsd/methodology/m49/

> part b - Run the code below to create a new migration flows dataset with the 10 countries you have chosen.

Solution:

```
# Change these countries to ones of interest to you!

# Really, change the countries
mysubset <- c("GRC", "TUR", "SYR", "LBY", "ITA", "MKD", "USA",
              "EGY", "DEU", "MAR")

migration_sub <- migration_flows_choice %>%
 filter(destcode %in% mysubset,
   origincode %in% mysubset)
```

> part c - Follow the steps in the code above to create a similar visualization but just for the 10 countries you selected, using only the minimal code you need to accomplish the task (e.g., you don't need to count edges).

Solution:

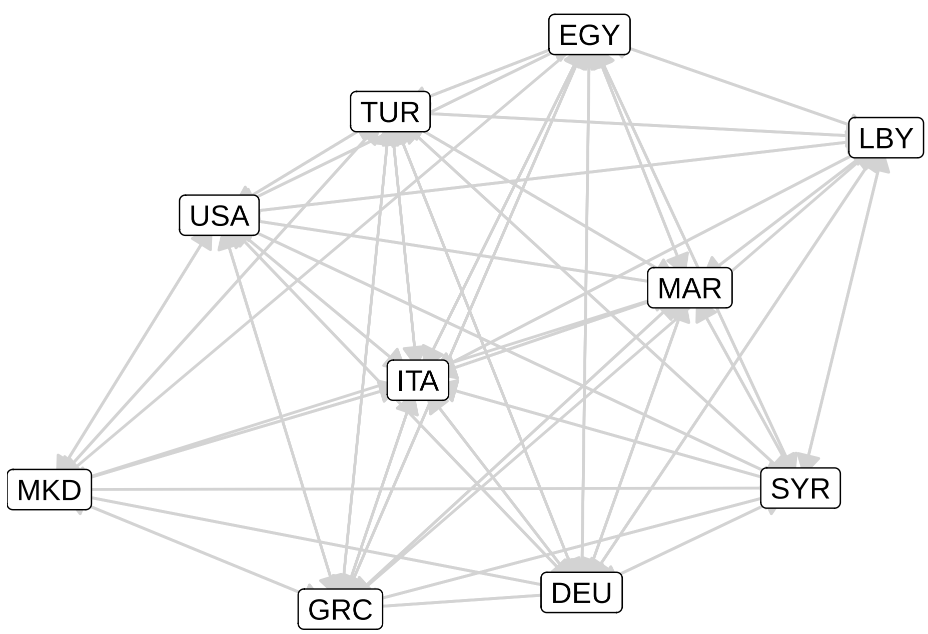
```r
# you need to construct the network before you can plot it
# call the igraph version migration_sub_igraph
# call the ggnetwork version migration_sub_network
migration_sub_igraph <- graph_from_data_frame(migration_sub,
           directed = TRUE)
summary(migration_igraph)
```

```
IGRAPH ccb31d7 DN-- 226 13805 --
+ attr: name (v/c), Y2000 (e/n)
```

```r
migration_sub_network <- ggnetwork(migration_sub_igraph)
head(migration_network)
```

```
          x         y name      xend      yend Y2000
1 0.0000000 0.2592951  NFK 0.5830937 0.4386270    91
2 0.0000000 0.2592951  NFK 0.6341176 0.5456436     2
3 0.2679849 0.7949914  MDV 0.6239215 0.5849770     2
4 0.2679849 0.7949914  MDV 0.5287109 0.6684206     1
5 0.2679849 0.7949914  MDV 0.6245632 0.6583652     5
6 0.2679849 0.7949914  MDV 0.6577961 0.5185068    11
```

```r
ggplot(data = migration_sub_network, aes(x = x, y = y,
          xend = xend, yend = yend)) +
 geom_edges(arrow = arrow(type = "closed", length = unit(8, "pt")),
    color = "lightgray") +
 geom_nodes() +
 geom_nodelabel(aes(label = name)) +
 theme_blank()
```



2 - Customizing the network graph

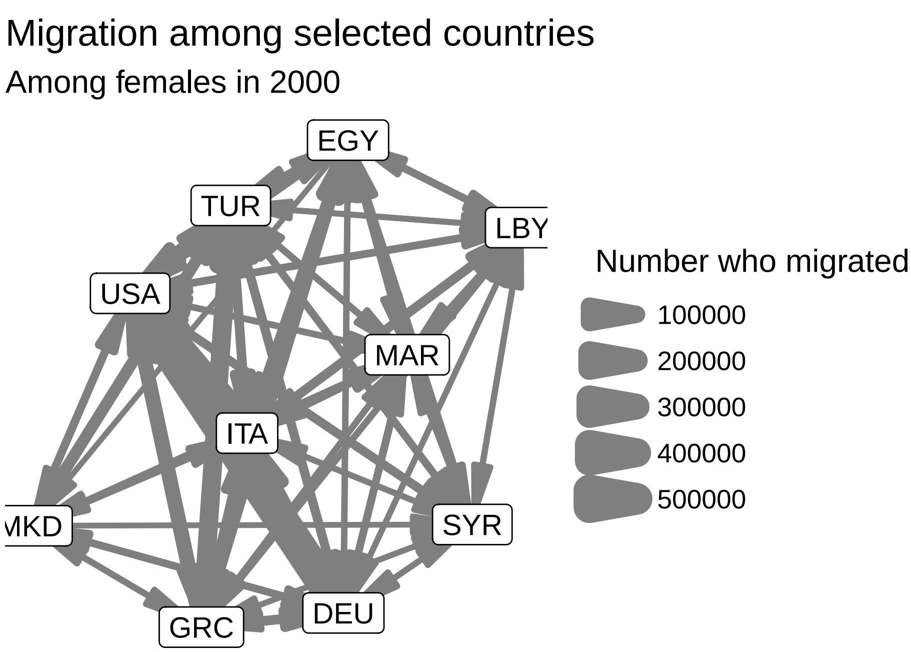
The plot of this network is much clearer than a plot of the entire network. Let's see how we can customize the network graph further.

> part a - Recalling that `Y2000` represents female migration in 2000, is this an edge or vertex attribute? (Adjust question in your mind if you choose differently!)

Solution: This would be an edge attribute.

Let's modify the graph so that edge width is a function of migration flow size. In `ggplot()` we can do this using the `size` option in `geom_edges()`.

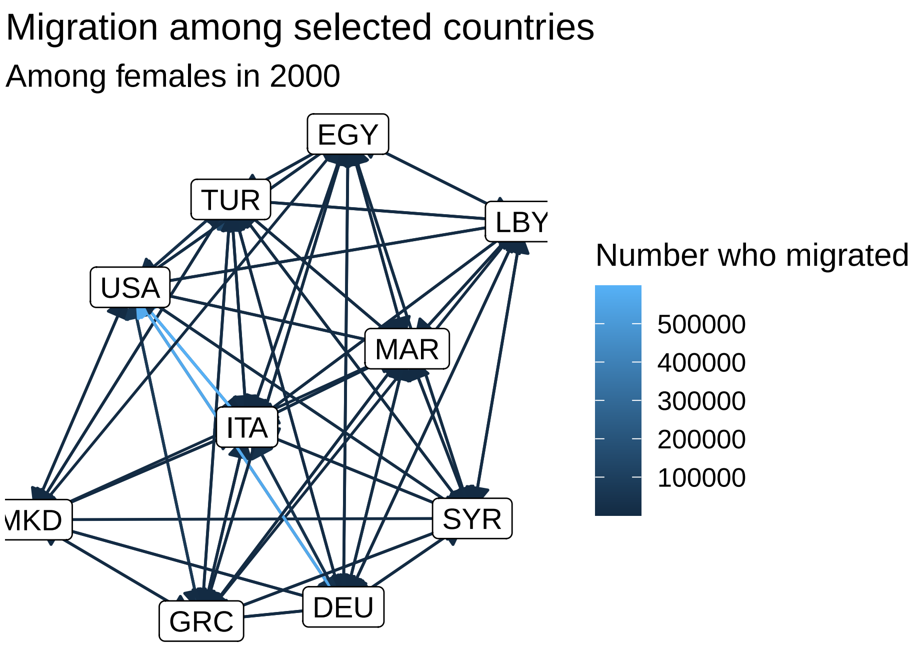
```
# assumes you called the network migration_sub_network
# change year in code and subtitle to whatever you chose
ggplot(data = migration_sub_network,
    aes(x = x, y = y,
      xend = xend, yend = yend)) +
 geom_edges(arrow = arrow(type = "closed", angle = 10),
       color = "gray50",
       aes(size = Y2000)) +
 geom_nodelabel(aes(label = name)) +
 labs(title = "Migration among selected countries",
    subtitle = "Among females in 2000",
    size = "Number who migrated") +
 theme_blank()
```



> part b - We could, instead, map edge color to the migration flow size. Which do you think is the more effective visual cue in this case?

Solution: In this case I think the first solution that adjusts the thickness of the lines is more effective, the colors in this case are too similar and it's harder to see the overall trends.

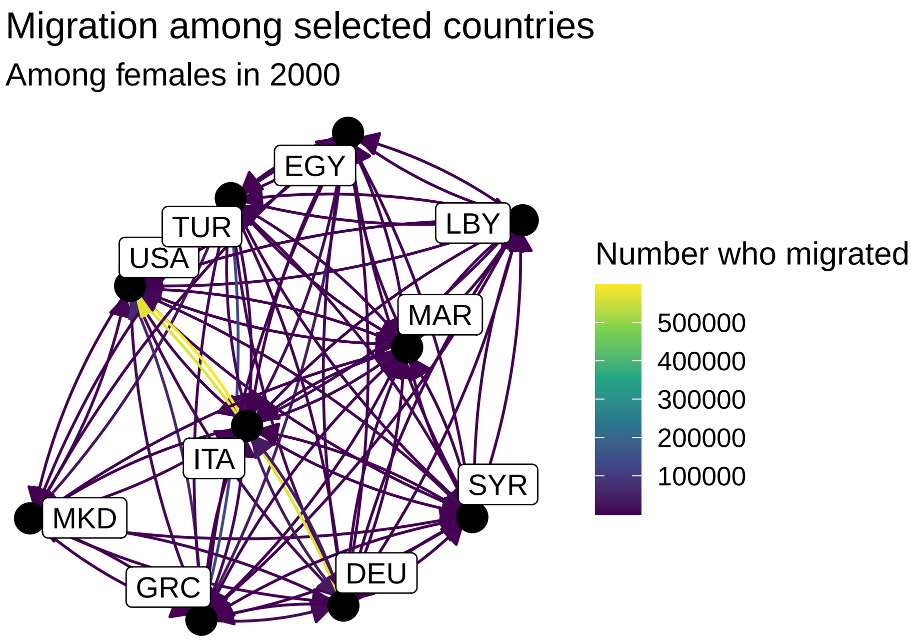
```r
# Adjust based on your choices again
ggplot(data = migration_sub_network,
    aes(x = x, y = y,
      xend = xend, yend = yend)) +
 geom_edges(arrow = arrow(type = "closed", length = unit(8, "pt")),
      aes(color = Y2000)) +
 geom_nodelabel(aes(label = name)) +
 labs(title = "Migration among selected countries",
    subtitle = "Among females in 2000",
    color = "Number who migrated") +
 theme_blank()
```



part c- Run the code below to see the same plot with a different color scheme. Is this more or less effective (or about the same)?

Solution: In my case this solution is about the same, but I would guess that in general it would be less effective because there's many more colors and so more convoluted to follow.

```r
ggplot(data = migration_sub_network,
    aes(x = x, y = y,
      xend = xend, yend = yend)) +
 geom_edges(arrow = arrow(type = "closed", length = unit(8, "pt")),
       curvature = 0.1,
       aes(color = Y2000)) +
 scale_color_continuous(type = "viridis") +
 geom_nodes(size = 5) +
 geom_nodelabel_repel(aes(label = name)) +
 labs(title = "Migration among selected countries",
    subtitle = "Among females in 2000",
    color = "Number who migrated") +
 theme_blank()
```


Migration among selected countries
Among females in 2000
EGY
TUR
LBY
USA
MAR
ITA
SYR
MKD
DEU
GRC
Number who migrated
500000
400000
300000
200000
100000

3 - Network centrality statistics

Let's consider some centrality statistics for the migration network of your chosen countries. We'll use `degree()` and `strength()` from the **igraph** package for this.

> part a - Based on *degree centrality*, which country(countries) were most central to the migration network of your chosen countries in 2000? Does the answer differ depending on whether we consider all edges (total degree), or only outgoing edges (out-degree; how many destinations were there from that origin country?) or only incoming edges (in-degree; how many origins were there to that destination country?)?

Solution: The most central countries to the migration network were Germany, Italy, Turkey, and the USA. When you look only at outgoing edges, this widens to include Turkey and Greece, and when you look only at incoming edges it widens to include Turkey and Syria.

```
igraph::degree(migration_sub_igraph)
```

```
DEU GRC ITA LBY MAR SYR TUR USA EGY MKD
 18  17  18  15  17  17  18  18  17  13
```

```
igraph::degree(migration_sub_igraph, mode = "out")
```

```
DEU GRC ITA LBY MAR SYR TUR USA EGY MKD
  9   9   9   7   9   8   9   9   9   6
```

```
igraph::degree(migration_sub_igraph, mode = "in")
```

```
DEU GRC ITA LBY MAR SYR TUR USA EGY MKD
  9   8   9   8   8   9   9   9   8   7
```

> part b - The `degree()` function only counts the number of edges of each node, but it does not account for the varying weights of those edges. We can use the `strength()` function to compute the weighted degrees instead. Do the same countries stand out as having high degree centrality after considering the weighted edges?

Solution: After considering weighted edges, the USA clearly stands out as having the highest degree centrality, followed by Italy and Germany. However, when considering only outgoing edges, Germany and Italy are highest, while when considering oncoming edges, the USA and Turkey are the highest.

```
# Get edge weights
migration_edge_weights <- edge_attr(migration_sub_igraph, name = "Y2000")

# Total movement
strength(migration_sub_igraph, weights = migration_edge_weights)
```

```
    DEU     GRC     ITA     LBY     MAR     SYR     TUR     USA     EGY     MKD
 682157  266250  730835   18316   19379   20104  218931 1302181   70513   38706
```

```r
# Total movement out
strength(migration_sub_igraph, weights = migration_edge_weights, mode = "out")
```

```
##    DEU    GRC    ITA    LBY    MAR    SYR    TUR    USA    EGY    MKD
## 612089 252128 667501    866  11462  16114  51866  23897  12307  35456
```

```r
# Total movement in
strength(migration_sub_igraph, weights = migration_edge_weights, mode = "in")
```

```
##     DEU     GRC     ITA     LBY     MAR     SYR     TUR     USA     EGY     MKD
##   70068   14122   63334   17450    7917    3990  167065 1278284   58206    3250
```

4 - Network of Thrones

Consider the data described in the article, *Network of Thrones* (Beveridge and Shan, 2017).

George R.R. Martin's fantasy novel, *A Storm of Swords*, was first published in 2000. About 13 years later, the first half of the novel was adapted for television in the third season of HBO's *Game of Thrones* (GoT). Our dataset is based on character interactions in the novel. Two characters are connected if their names appear within 15 words of one another in the novel. The dataset provides the edge lists and weights from the novel. The edge weight counts the number of these occurrences. The edge list is not directed (even though the variables names suggest such).

```
got <- read_csv("data/storm-of-swords.csv")
```

> part a - Think about the text as data: Suppose, instead of the formatted data above, we had the entire text of the novel. List some of the steps (in English or pseudocode) required to wrangle the data into the form above.

Solution: First, we would need to find the location of all the occurrences of the names we are interested in examining. Next, we would need to

> part b - How many GoT characters (nodes) and character interactions (edges) are in this network?

Solution: There are 107 nodes and 352 edges.

```
# Create igraph object called got_igraph
got_igraph <- graph_from_data_frame(got,
           directed = FALSE)
summary(got_igraph)
```

```
IGRAPH ce6ea73 UN-- 107 352 --
+ attr: name (v/c), Weight (e/n)
```

```
# Identify number of nodes and edges
# Get descriptions and counts of vertices
V(got_igraph) # not necessarily useful to print
```

```
+ 107/107 vertices, named, from ce6ea73:
  [1] Aemon       Aerys       Alliser     Amory       Arya
  [6] Balon       Belwas      Beric       Bran        Brienne
 [11] Bronn       Brynden     Catelyn     Cersei      Craster
 [16] Daario      Daenerys    Davos       Eddard      Eddison
 [21] Edmure      Gendry      Gilly       Gregor      Hodor
 [26] Hoster      Irri        Jaime       Janos       Joffrey
 [31] Jojen       Jon         Jon Arryn   Jorah       Kevan
 [36] Loras       Lothar      Luwin       Lysa        Mance
 [41] Meera       Melisandre  Meryn       Missandei   Myrcella
 [46] Oberyn      Podrick     Rattleshirt Renly       Rhaegar
+ ... omitted several vertices
```

```r
vcount(got_igraph)
```

```
[1] 107
```

```r
# Get descriptions and counts of edges
E(got_igraph) # not necessarily useful to print
```

```
+ 352/352 edges from ce6ea73 (vertex names):
 [1] Aemon  --Grenn     Aemon  --Samwell   Aerys  --Jaime     Aerys  --Robert
 [5] Aerys  --Tyrion    Aerys  --Tywin     Alliser--Mance     Amory  --Oberyn
 [9] Arya   --Anguy     Arya   --Beric     Arya   --Bran      Arya   --Brynden
[13] Arya   --Cersei    Arya   --Gendry    Arya   --Gregor    Arya   --Jaime
[17] Arya   --Joffrey   Arya   --Jon       Arya   --Rickon    Arya   --Robert
[21] Arya   --Roose     Arya   --Sandor    Arya   --Thoros    Arya   --Tyrion
[25] Balon  --Loras     Belwas --Barristan Belwas --Illyrio   Beric  --Anguy
[29] Beric  --Gendry    Beric  --Thoros    Bran   --Hodor     Bran   --Jojen
[33] Bran   --Jon       Bran   --Luwin     Bran   --Meera     Bran   --Nan
[37] Bran   --Rickon    Bran   --Samwell   Bran   --Theon     Brienne--Loras
+ ... omitted several edges
```

```r
ecount(got_igraph)
```

```
[1] 352
```

part c - What proportion of possible edges are realized?

This proportion is referred to as the "density" of a graph, which is a measure of how close the number of observed edges are to the maximal possible number of edges. Density ranges from 0 (least dense or sparser) to 1 (most dense) and can be obtained with the `edge_density()` function from **igraph**. Use this function to get the density, and verify it's correct by calculating the density yourself.

Note: The number of *possible* edges in an undirected graph is $\binom{V}{2} = \frac{V(V-1)}{2}$.

Solution: The density of this graph is .031, indicating that it is not very dense at all compared to the proportion of possible edges that there could be.

```r
igraph::edge_density(got_igraph)
```

```
[1] 0.06207018
```

part d - The function `is_connected()` returns "TRUE" if a graph is connected and "FALSE" otherwise. Is this graph connected? And if so, what does that mean? How would you be able to tell that the graph was connected by looking at Figure 2 in the *Network of Thrones* paper?

Solution:

part e - Use the code below to compute the diameter of the network. Interpret the value.

Solution:

```r
diameter(got_igraph, directed = FALSE)
```

```
[1] 6
```

5 - Network of Thrones: Centrality statistics

Next, let's consider the centrality statistics for characters in the network. The node degree counts the number of characters that a given character (node) is associated with. The weighted degree (given by `strength()`) is the sum of the edge weights for edges connecting one character (node) to other characters. In other words, the strength counts the total number of interactions a character has with others in the network. Below, we compute the degree and strength of each node, and combine these vectors into a dataframe.

```
got_stats <- data.frame(name = vertex_attr(got_igraph, "name"),
            degree = degree(got_igraph),
            strength = strength(got_igraph,
                      weights = edge_attr(got_igraph, "Weight")))
```

> part a - Who are the five characters with highest degree? Highest weighted degree? Verify that these values (look like they) match those in Figure 3 of the *Network of Thrones* paper.

Solution:

> part b - Explain how Robb can have higher degree than Bran but lower weighted degree.

You can answer this without knowing any of the GoT story.

Solution:

```
# may not need this chunk
```

> part c - Now consider the (unweighted) betweenness measure of centrality. In the code below, we use the `betweenness()` function to calculate the unweighted betweenness of the nodes, and add this statistics to the `got_stats` data frame using `add_column()`. Verify that the top ranked characters match those shown in Figure 3 of the *Network of Thrones* paper.

Solution:

```
got_stats <- got_stats %>%
 add_column(betweenness = betweenness(got_igraph, weights = NA))
```

Lastly, let's consider eigenvector centrality and Google PageRank. The *Network of Thrones* paper gives a simple description of the page rank centrality measure. The basic idea is that a node will have a higher page rank value (and higher "centrality") if it is connected to important nodes. The page rank of node i is a function of the weighted sum of the page ranks of its neighbors (who i is connected to) with weights given by the edge weight between node i and its neighbor, divided by the total weighted degree of the neighbor.

Example: Consider the page ranks of Catelyn and Hodor. Both are connected to Bran, who has a weighted degree of 344. Bran has a total of 4 interactions with Catelyn so his page rank value is weighted by the fraction 4/344, or 0.01, when computing Catelyn's page rank. But Hodor's page rank calculation is influenced much more by Bran's value, since he has 96 interactions with Bran, which makes up a 96/344, or 0.28, fraction of all of Bran's interactions. In this way, Hodor's page rank will be closer to Bran's value because he has more interactions with him than Catelyn.

> part d - Use the provided code to add two variables to the `got_stats` dataframe: one with the unweighted eigenvector centrality, and a second with the unweighted page rank. Which characters score in the top 5 according to the page rank measure?

Solution:

```
got_stats <- got_stats %>%
 add_column(eigen = eigen_centrality(got_igraph, directed = FALSE,
                                     weights = NA) %>%
              purrr::pluck("vector"),
       pagerank = page_rank(got_igraph, directed = FALSE,
                            weights = NA) %>%
         purrr::pluck("vector"))
```

part e - How can a character like Daenerys have such a high page rank, and a high rank for betweenness, but a low degree? (You can use Figure 2 in the *Network of Thrones* paper to visualize the structure.)

```
# you may or may not need the code chunk
```

part f - Finally, plot the network with node or label size determined by the page rank value.

When plotting the graph, will it look better with igraph or ggnetwork being used? Use what will look better.

Solution:

```
# Add page rank as a vertex attribute


# Graph network
```

6 - Community detection

Community detection in networks is a process of finding clusters of nodes (communities) that are highly connected within a cluster and have few connections across clusters. In other words, this is clustering, but as mentioned in your prep, the methods are very different.

Figure 2 in the *Network of Thrones* paper uses color to denote the 7 communities found in their analysis. There are a variety of algorithms to do this, but most depend on calculating the modularity of the cluster assignment, which is a measure of how well a network can be divided into clusters. Modularity compares the edge weight between two nodes in the same cluster to the expected weight between the two nodes in a graph with a random assignment of edges. The higher the modularity value, the better the division into clusters (with a max value of 1).

In *Network of Thrones*, the authors use the Louvain algorithm, which is a hierarchical method similar to hierarchical clustering for unsupervised learning. Nodes start out as individual clusters, then are merged together to create communities to increase modularity the most at each step (in a local, greedy way). The algorithm stops when the modularity value can't be increased by an additional step.

> part a - Run the code below to implement Louvain clustering and compute the modularity. What value did you obtain?

Solution:

```r
# Identify clusters using Louvain algorithm
got_cl <- cluster_louvain(got_igraph)
got_cl
```

```
IGRAPH clustering multi level, groups: 6, mod: 0.49
+ groups:
  $`1`
   [1] "Aemon"       "Alliser"     "Craster"     "Davos"       "Eddison"
   [6] "Gilly"       "Janos"       "Jon"         "Mance"       "Melisandre"
  [11] "Rattleshirt" "Samwell"     "Shireen"     "Stannis"     "Val"
  [16] "Ygritte"     "Grenn"       "Karl"        "Cressen"     "Salladhor"
  [21] "Bowen"       "Dalla"       "Orell"       "Qhorin"      "Styr"

  $`2`
   [1] "Aerys"        "Amory"        "Balon"        "Bronn"
   [5] "Cersei"       "Gregor"       "Jaime"        "Joffrey"
  + ... omitted several groups/vertices
```

```r
# Compute modularity from Louvain clustering
modularity(got_cl)
```

```
[1] 0.4895806
```

> part b - After clustering, we can determine how many nodes are in each detected cluster (i.e., how many characters are in each detected community). How many communities are there, and how many characters are there in each community?

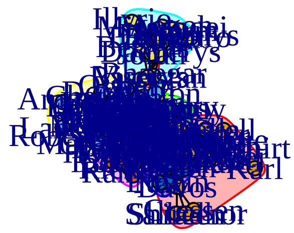
Solution:

```
communities(got_cl)
```

```
$`1`
 [1] "Aemon"       "Alliser"     "Craster"     "Davos"       "Eddison"
 [6] "Gilly"       "Janos"       "Jon"         "Mance"       "Melisandre"
[11] "Rattleshirt" "Samwell"     "Shireen"     "Stannis"     "Val"
[16] "Ygritte"     "Grenn"       "Karl"        "Cressen"     "Salladhor"
[21] "Bowen"       "Dalla"       "Orell"       "Qhorin"      "Styr"

$`2`
 [1] "Aerys"        "Amory"        "Balon"        "Bronn"        "Cersei"
 [6] "Gregor"       "Jaime"        "Joffrey"      "Jon Arryn"    "Kevan"
[11] "Loras"        "Lysa"         "Meryn"        "Myrcella"     "Oberyn"
[16] "Podrick"      "Renly"        "Robert Arryn" "Sansa"        "Shae"
[21] "Tommen"       "Tyrion"       "Tywin"        "Varys"        "Walton"
[26] "Elia"         "Ilyn"         "Pycelle"      "Qyburn"       "Margaery"
[31] "Lancel"       "Olenna"       "Marillion"    "Ellaria"      "Mace"
[36] "Chataya"      "Doran"

$`3`
[1] "Arya"   "Beric"  "Eddard" "Gendry" "Robert" "Sandor" "Anguy"  "Thoros"

$`4`
 [1] "Belwas"    "Daario"    "Daenerys"  "Irri"      "Jorah"     "Missandei"
 [7] "Rhaegar"   "Viserys"   "Barristan" "Illyrio"   "Drogo"     "Aegon"
[13] "Kraznys"   "Rakharo"   "Worm"

$`5`
[1] "Bran"   "Hodor"  "Jojen"  "Luwin"  "Meera"  "Rickon" "Nan"    "Theon"

$`6`
 [1] "Brienne" "Brynden" "Catelyn" "Edmure"  "Hoster"  "Lothar"  "Rickard"
 [8] "Robb"    "Roose"   "Walder"  "Jeyne"   "Petyr"   "Roslin"  "Ramsay"
```

As we saw in the prep, you can plot the network with the following code, but this graph is harder to customize.

```
plot(got_cl, got_igraph)
```



part c - Create a better plot of the network with `ggplot()`, and color by group membership.

Solution:

```
# Get community membership
got_membership <- membership(got_cl)

# Add community membership as vertex attribute


# Create a plot
```