Lab 8 - Networks - Example Solution

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.

```
# 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`. (Again, update to whatever you want to examine!)

How many rows are in your final dataset?

The default has 13805 rows (will change if you change the year, etc.).

```
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).

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

```
IGRAPH a835398 DN-- 226 13805 --
+ attr: name (v/c), Y2000 (e/n)
```

Then we can get basic statistics about the network.

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

```
+ 226/226 vertices, named, from a835398:
  [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
```

```
vcount(migration_igraph)
```

```
[1] 226
```

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

```
+ 13805/13805 edges from a835398 (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
```

```
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 vertices/nodes and 13805 edges in the network.

```
# 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.

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

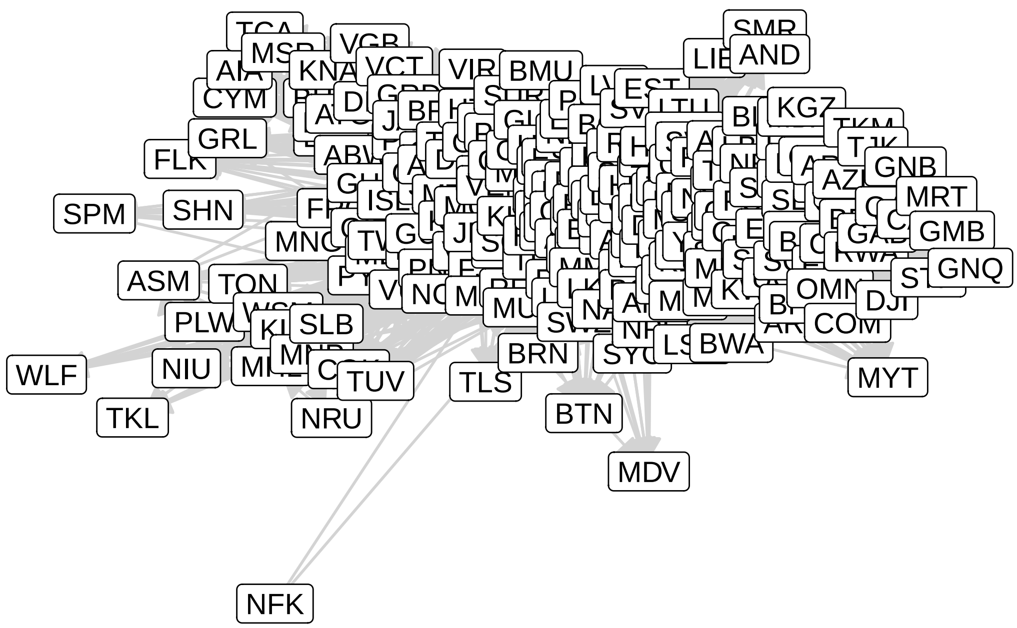
While this can work reasonably well for small graphs, we can create a better visualization of this network using `ggnetwork()` to convert the igraph object to a network object, and `ggplot()` to plot the network graph.

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

```
  x         y name      xend      yend Y2000
1 0 0.3987415  WLF 0.5451186 0.6730128     1
2 0 0.3987415  WLF 0.2789369 0.4804065     4
3 0 0.3987415  WLF 0.2276603 0.4978025     4
4 0 0.3987415  WLF 0.3227670 0.5603428    10
5 0 0.3987415  WLF 0.3884132 0.6054213     2
```

```
  6 0 0.3987415  WLF 0.4022982 0.5319155    19
```

```r
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:

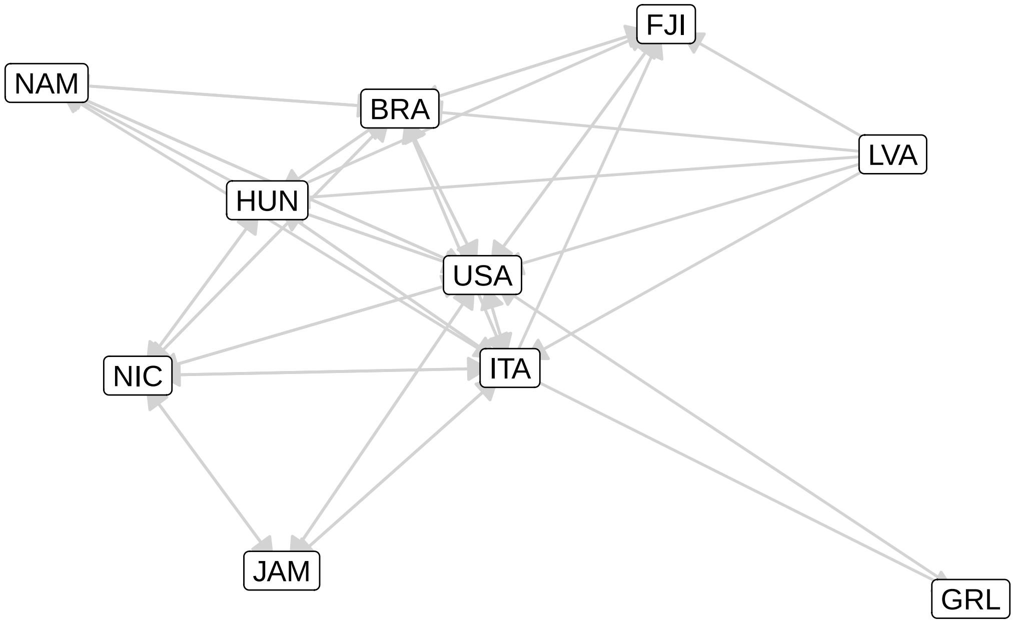
The countries I chose (yours should be different!) are: USA, Brazil, Namibia, Latvia, Italy, Jamaica, Hungary, Greenland, Fiji, and Nicaragua.

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

# Really, change the countries
mysubset <- c("BRA", "USA", "NAM", "LVA", "ITA", "JAM", "HUN",
              "GRL", "FJI", "NIC")

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:

```
# 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)
migration_sub_network <- migration_sub_igraph %>% ggnetwork()

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()
```


FJI
NAM
BRA
LVA
HUN
USA
NIC
ITA
JAM
GRL

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.

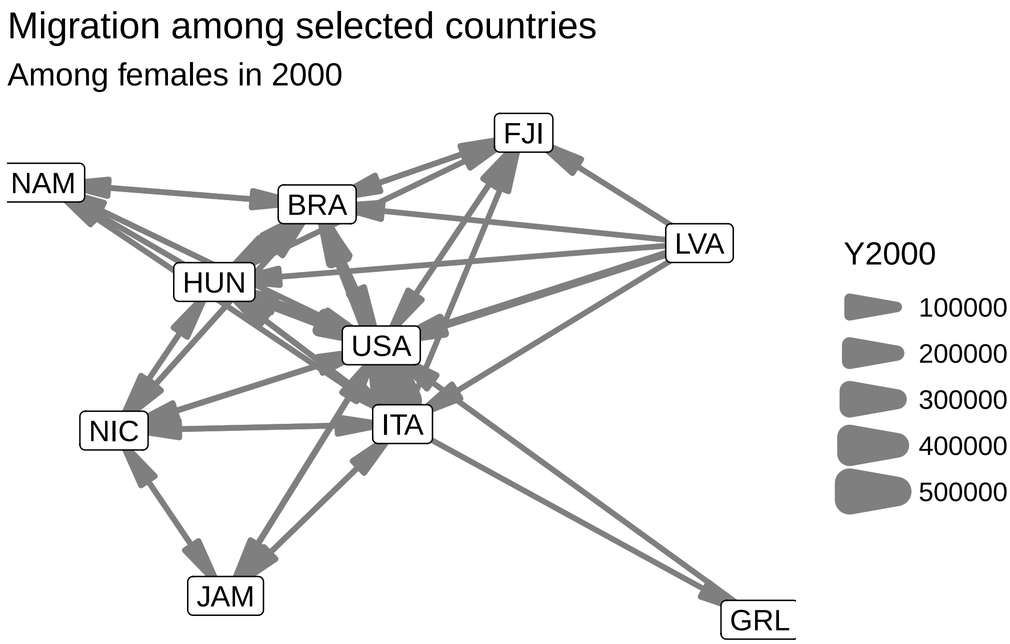
> part a - Recalling that `Y2000` represents female migration in 2000, is this an edge or vertex attribute? (Adjust question in your mind if you choose the year or male/female differently!)

Solution:

This is an edge attribute. (We accessed it that way above.)

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()`.

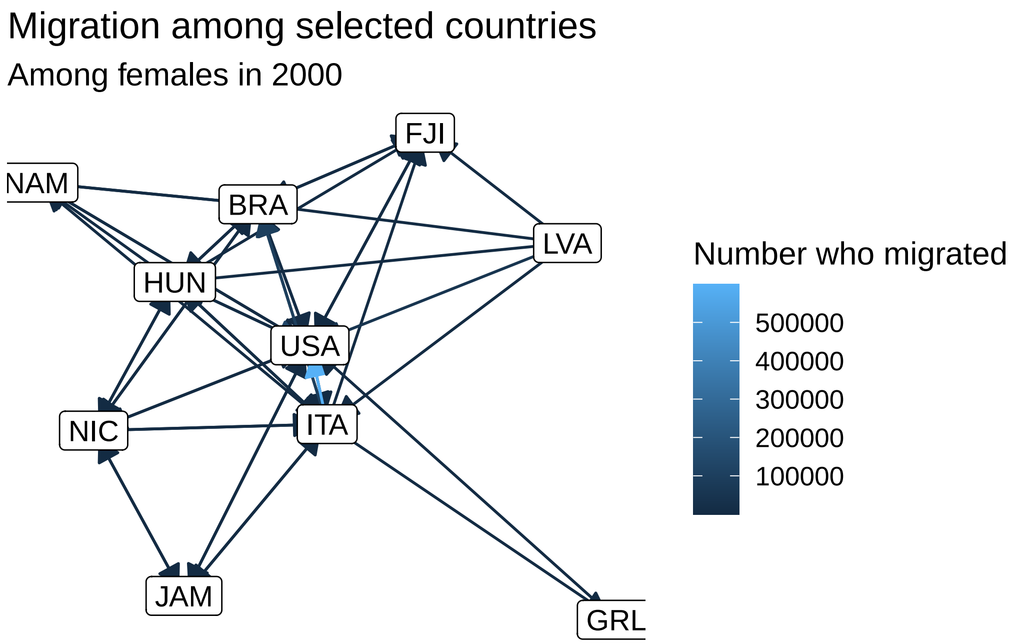
```
# assumes you called the network migration_sub_network
# change year in code and subtitle to whatever you chose
ggplot(data = migration_sub_network,
    aes(x = x, y = y,
      xend = xend, yend = yend)) +
 geom_edges(arrow = arrow(type = "closed", angle = 10),
       color = "gray50",
       aes(linewidth = 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:

```
# 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()
```

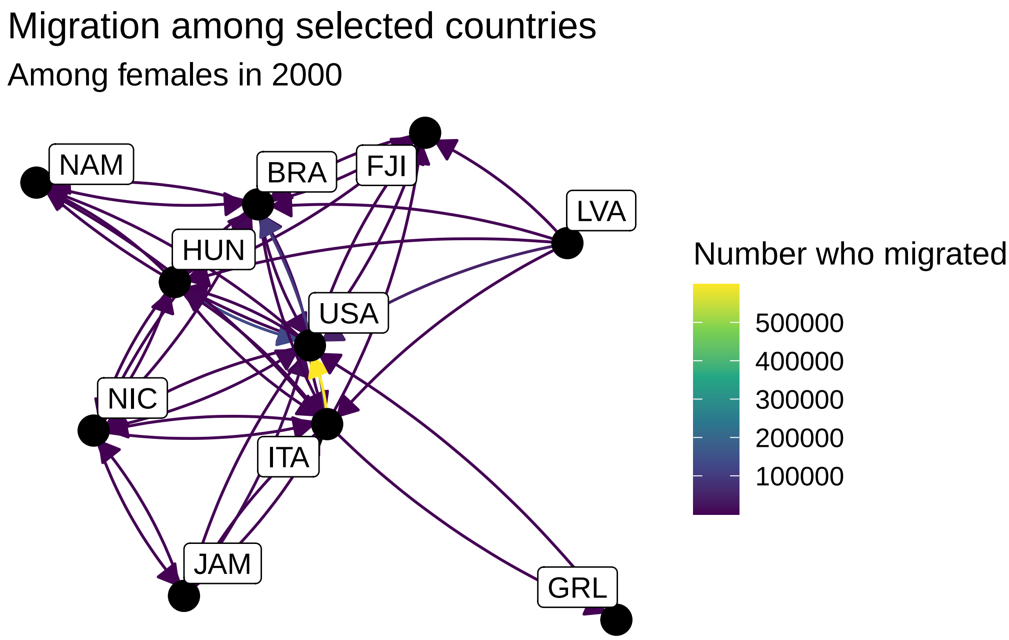


Edge size appears to be the more effective visual cue for seeing differences in migration flow. It's too hard to see the color differences in my opinion.

> 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:

```
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()
```



This color scheme may help to distinguish the migration patterns a little better. In my set of countries, most values are very small, with one arrow into the USA being very high, so it is hard to distinguish all the low ones from each other. Note that in the plot, there are two other things I did to make the arrow direction more visible:

1. I added curvature to the arrows so that arrows going to and from a country were not right on top of each other (these were overlapping when there was no curvature, making it harder to distinguish colors).

2. To better see the arrowheads, I added `geom_nodes()` to add large points to the nodes, and I used `geom_nodelabel_repel()` instead of `geom_nodelabel` to move the node labels off the nodes slightly.

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:

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

```
FJI HUN ITA LVA NAM NIC USA BRA JAM GRL
  7  11  14   5   6  10  16  13   6   2
```

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

```
FJI HUN ITA LVA NAM NIC USA BRA JAM GRL
  2   6   8   5   2   5   7   6   3   1
```

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

```
FJI HUN ITA LVA NAM NIC USA BRA JAM GRL
  5   5   6   0   4   5   9   7   3   1
```

Answers will vary based on your selected countries.

From this output, we can see that the USA, Italy (ITA), and Brazil (BRA) were the top 3 countries in terms of overall degree. These remain the top 3 in both in and out degree but in different orders. For example, Italy has the highest out degree, while the USA has the highest in degree. Hungary would be included in a tie for 3rd in terms of highest out degree.

> 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:

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

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

```
   FJI    HUN    ITA    LVA    NAM    NIC    USA    BRA    JAM    GRL 
  1144 144777 703712  49434    138   7138 816932 117474  12400     33 
```

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

```
   FJI    HUN    ITA    LVA    NAM    NIC    USA    BRA    JAM    GRL 
   537 140697 700956  49434     13   6406   9604   7541  11371     32 
```

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

```
   FJI    HUN    ITA    LVA    NAM    NIC    USA    BRA    JAM    GRL 
   607   4080   2756      0    125    732 807328 109933   1029      1 
```

Answers will vary based on the countries you chose.

Interestingly here, we see differences in countries "centrality" based on in versus out direction information. In particular, Italy and Hungary had a large exodus of people (Italy's is WAY higher than I would have imagined), and the USA and Brazil had large intakes. The US intake is by far the largest of these. I was surprised to see that only one female from Italy immigrated to Greenland in Y2000 (among the countries I picked).

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 that).

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

glimpse(got)
```

```
Rows: 352
Columns: 3
$ Source <chr> "Aemon", "Aemon", "Aerys", "Aerys", "Aerys", "Aerys", "Alliser"~
$ Target <chr> "Grenn", "Samwell", "Jaime", "Robert", "Tyrion", "Tywin", "Manc~
$ Weight <dbl> 5, 31, 18, 6, 5, 8, 5, 5, 11, 23, 9, 6, 5, 43, 7, 11, 6, 7, 8, ~
```

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:

Added glimpse above to see what the data looks like in order to answer this question.

Some steps would include…

- create a list that contains the names (and nicknames) of each character of interest
- identify each instance of a character's name (or nickname), and extract the word number within the novel, e.g.:
- use `unnest_tokens()` on the text
- take row number to be the word number
- filter for character names of interest such that the resulting dataset is one row per mention of any character (with variables for the character name and the word number)
- for each character, assign vector of word numbers at which point their character name appears in the text;
- search other character appearances within $pm15$ words
- group by other character and count the instances of appearances within ±15 words
- remove duplicate rows resulting from previous step

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

Solution:

There are 107 characters in this network and 352 character interactions (edges).

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

# Identify number of nodes and edges
summary(got_igraph)
```

```
IGRAPH a960821 UN-- 107 352 --
+ attr: name (v/c), Weight (e/n)
```

```
## Alternative
# ecount(got_igraph)
# vcount(got_igraph)
```

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:

There are 352 realized edges out of 5671 possible edges in this undirected graph. The graph density is thus 6.2%, representing a rather sparse network.

```
# use appropriate density command
edge_density(got_igraph)
```

```
[1] 0.06207018
```

```
# manual computation
ecount(got_igraph) / choose(vcount(got_igraph), 2)
```

```
[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:

A graph is called connected if there is a path between all pairs of vertices. That is, you can travel from any node to any other node. It also means that there is 1 component, not several. Here, that means that there is a path between all characters in the novel. We could tell this from Figure 2 in the GOT paper because (1) each node has at least one path to it; and (2) there are no disconnected groups (i.e., there is always at least one path between the different groups of nodes).

```
igraph::is_connected(got_igraph)
```

```
[1] TRUE
```

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

Solution:

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

```
[1] 6
```

The diameter of the network is 6, meaning that the longest geodesic (shortest path) between any two characters is 6-characters away. This echoes the sentiment conveyed in the phrase "6 degrees of separation" that you also saw in the textbook.

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:

```
# Highest degrees
got_stats %>%
 arrange(desc(degree)) %>%
 head(5)
```

```
         name degree strength
Tyrion Tyrion     36      551
Jon       Jon     26      442
Sansa   Sansa     26      383
Robb     Robb     25      342
Jaime   Jaime     24      372
```

```
# Highest strengths
got_stats %>%
 arrange(desc(strength)) %>%
 head(5)
```

```
         name degree strength
Tyrion Tyrion     36      551
Jon       Jon     26      442
Sansa   Sansa     26      383
```

```
Jaime    Jaime      24       372
Bran      Bran      14       344
```

Tyrion, Jon, Sansa, Robb, and Jaime are the five characters with the highest degree. Tyrion, Jon, Sansa, Jaime, and Bran are the five characters with the highest weighted degree. These results appear to match the values found in Figure 3 of the *Network of Thrones* paper.

> 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:

Degree just counts number of characters interacted with, and doesn't consider how often interactions occurred. If you interact with 30 people just once, and someone else interacts with 10 people but 10 times each, the first would have a higher degree but lower weighted degree than the latter.

Here, in particular, Robb has interactions with more characters than Bran (25 vs. 14), but Bran has more interactions with each of the characters he interacts with so has more interactions in total than Robb (344 vs. 342 - yes this is close but clearly Bran is interacting more with the 14 individuals on average than Robb was with his 25).

```
# used chunk to get statistics of interest
got_stats %>%
 filter(name %in% c("Bran", "Robb"))
```

```
     name degree strength
Bran Bran     14      344
Robb Robb     25      342
```

> 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))

got_stats %>%
```

```
  arrange(desc(betweenness)) %>%
  head(5)
```

```
               name degree strength betweenness
Jon             Jon     26      442   1279.7534
Robert       Robert     18      128   1165.6025
Tyrion       Tyrion     36      551   1101.3850
Daenerys Daenerys     14      232    874.8372
Robb           Robb     25      342    706.5573
```

Jon, Robert, Tyrion, Daenerys, and Robb are the top ranked characters according to the betweenness measure of centrality.

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"))

got_stats %>%
```

```r
arrange(desc(pagerank)) %>%
head(5)
```

```
               name degree strength betweenness      eigen   pagerank
Tyrion       Tyrion     36      551   1101.3850 1.00000000 0.04288498
Jon             Jon     26      442   1279.7534 0.42314218 0.03582870
Robb           Robb     25      342    706.5573 0.72731374 0.03017115
Sansa         Sansa     26      383    705.1986 0.82812833 0.03000972
Daenerys Daenerys     14      232    874.8372 0.07437362 0.02881425
```

Tyrion, Jon, Robb, Sansa, and Daenerys score in the top 5 according to the page rank measure.

> 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.)

Daenerys has (unweighted) degree 14, indicating she has direct connections to only 14 other characters. Thus, she interacts primarily with a small group of individuals and is a far distance (path) to others in the novel. However, she appears to be the only connection to many of the characters in the green cluster of Figure 2 (the shortest path to these characters needs to go through Daenerys), and thus she acts to connect these individuals to the wider network of characters (hence the relatively high betweenness). She gets all or most of the page rank influence of her close circle of friends, since she is their sole or majority weight connection in the network. Other characters, like Robb, may have higher degree or weighted degree, but their associates also have a high degree so the proportion of their page rank that goes to Robb is small.

```r
got_stats %>%
 filter(name %in% c("Daenerys"))
```

```
               name degree strength betweenness      eigen   pagerank
Daenerys Daenerys     14      232    874.8372 0.07437362 0.02881425
```

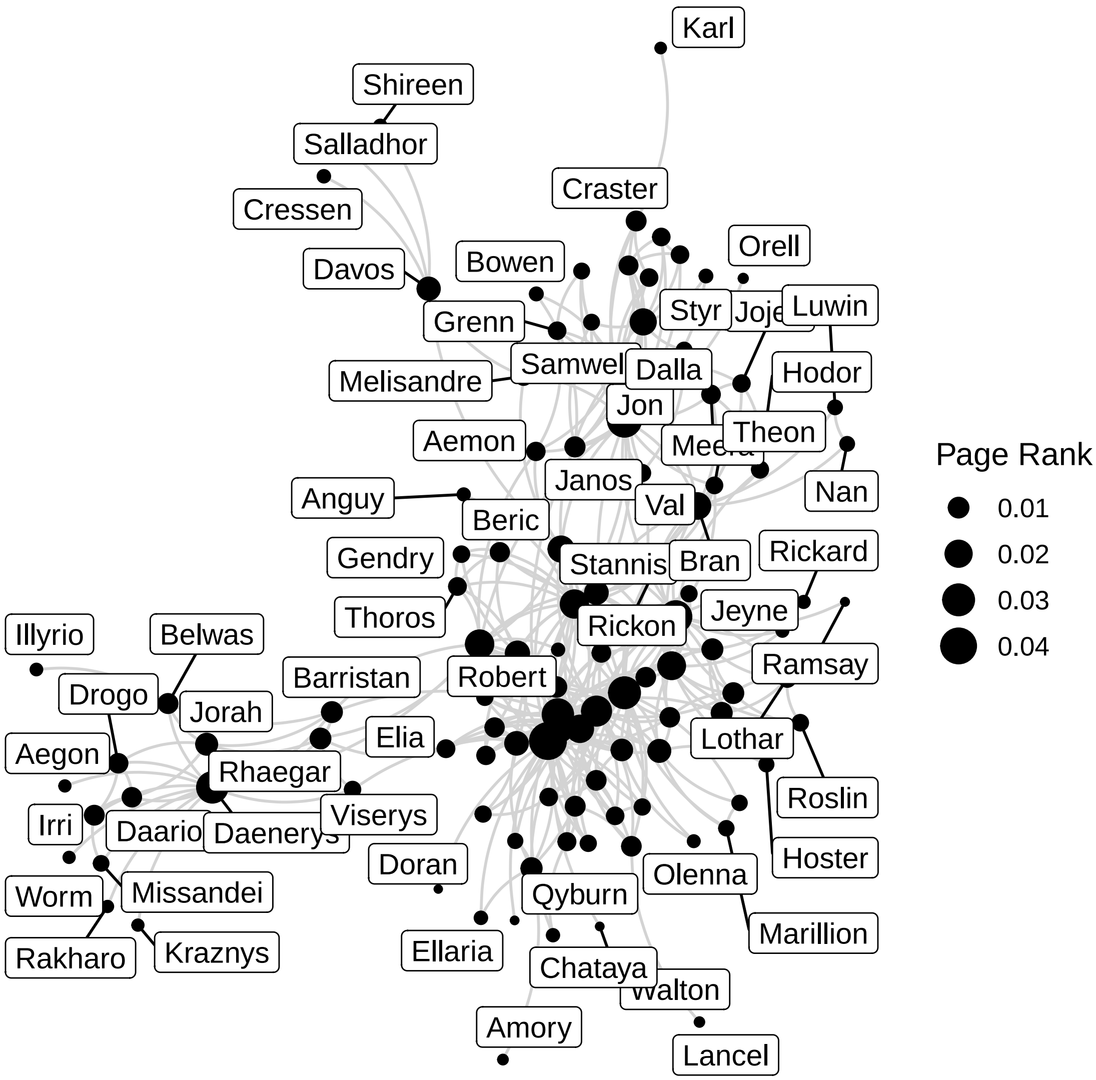
> part f - Finally, plot the network with node or label size determined by the page rank value.

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

Solution:

```r
# Add page rank as a vertex attribute
got_igraph <- set_vertex_attr(got_igraph, name = "pagerank", value = got_stats$pagerank)

# Make into ggnetwork object
got_network <- ggnetwork(got_igraph)
```

```r
set.seed(231)

# Graph network
ggplot(data = got_network,
        aes(x = x, y = y, xend = xend, yend = yend)) +
 geom_edges(color = "lightgray", curvature = 0.2) +
  # for whatever reason, this will not compile without the $ syntax
  # which it actively discourages
 geom_nodes(aes(size = got_network$pagerank)) +
 geom_nodelabel_repel(aes(label = name)) +
 labs(size = "Page Rank") +
 theme_blank()
```



Karl
Shireen
Salladhor
Cressen
Craster
Orell
Davos
Bowen
Styr
Joje
Luwin
Grenn
Samwel
Dalla
Hodor
Melisandre
Jon
Theon
Aemon
Mee
Nan
Janos
Val
Anguy
Beric
Gendry
Stannis
Bran
Rickard
Thoros
Jeyne
Rickon
Illyrio
Belwas
Ramsay
Barristan
Robert
Drogo
Jorah
Elia
Lothar
Aegon
Rhaegar
Roslin
Viserys
Irri
Daario
Daenerys
Doran
Hoster
Olenna
Worm
Missandei
Qyburn
Marillion
Rakharo
Kraznys
Ellaria
Chataya
Walton
Amory
Lancel
Page Rank
0.01
0.02
0.03
0.04

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. There are other community detection algorithms based on a partitioning approach, like in k-means clustering.

> 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"    "Cersei"   "Gregor"
   [7] "Jaime"    "Joffrey"  "Kevan"    "Loras"    "Meryn"    "Myrcella"
  + ... omitted several groups/vertices
```

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

```
[1] 0.4904442
```

The modularity is 0.4896. This is actually pretty high (considering my experience with networks in the past).

> 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:

```r
communities(got_cl)
```

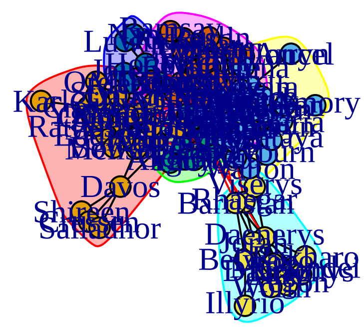
```
$`1`
 [1] "Aemon"       "Alliser"     "Craster"     "Davos"       "Eddison"
 [6] "Gilly"       "Janos"       "Jon"         "Mance"       "Melisandre"
[11] "Rattleshirt" "Samwell"     "Shireen"     "Stannis"     "Val"
[16] "Ygritte"     "Grenn"       "Karl"        "Cressen"     "Salladhor"
[21] "Bowen"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

$`2`
 [1] "Aerys"    "Amory"    "Balon"    "Bronn"    "Cersei"   "Gregor"
 [7] "Jaime"    "Joffrey"  "Kevan"    "Loras"    "Meryn"    "Myrcella"
[13] "Oberyn"   "Podrick"  "Renly"    "Shae"     "Tommen"   "Tyrion"
[19] "Tywin"    "Varys"    "Walton"   "Elia"     "Ilyn"     "Pycelle"
[25] "Qyburn"   "Margaery" "Lancel"   "Olenna"   "Ellaria"  "Mace"
[31] "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" "Nan"
```

```
$`6`
 [1] "Brienne"      "Brynden"      "Catelyn"      "Edmure"       "Hoster"
 [6] "Jon Arryn"    "Lothar"       "Lysa"         "Rickard"      "Rickon"
[11] "Robb"         "Robert Arryn" "Roose"        "Sansa"        "Walder"
[16] "Theon"        "Jeyne"        "Petyr"        "Roslin"       "Marillion"
[21] "Ramsay"
```

There are 6 communities. The 1st has 25 characters, 2nd has 37, third has only 8, fourth has 15, 5th has only 8, and finally the 6th has 14.

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

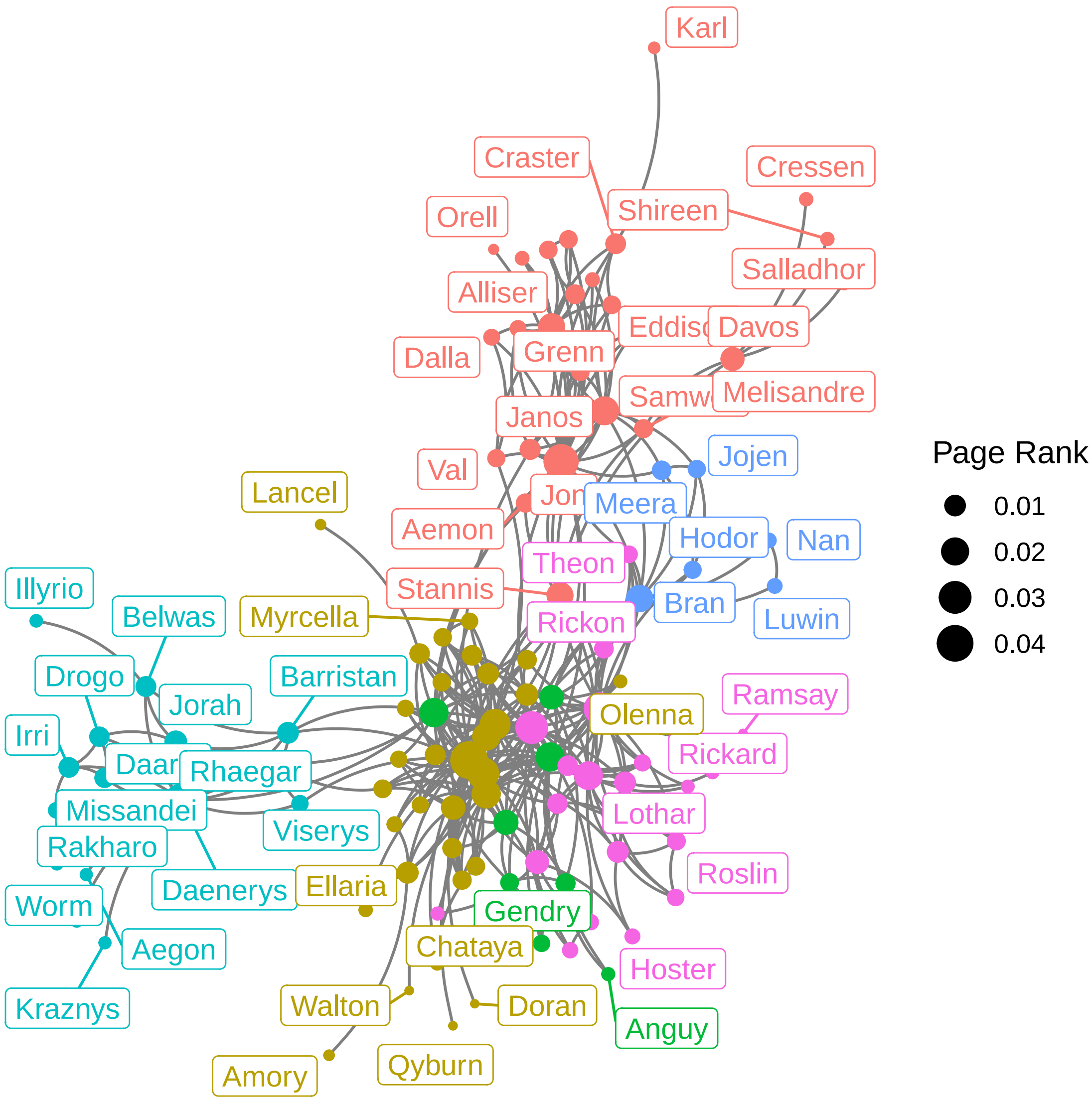
```r
plot(got_cl, got_igraph)
```



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
got_igraph <- set_vertex_attr(got_igraph,
                              name = "membership",
                              value = got_membership)

got_network <- ggnetwork(got_igraph) %>%
              mutate(membership = factor(membership))

# Create a plot
set.seed(231) # may not be needed
got_network %>%
  ggplot(aes(x = x, y = y, xend = xend, yend = yend)) +
  geom_edges(color = "gray50", curvature = 0.2) +
   # for whatever reason, this will not compile without the $ syntax
  # which it actively discourages
  geom_nodes(aes(size = got_network$pagerank, color = membership)) +
  geom_nodelabel_repel(aes(label = name, color = membership)) +
  labs(size = "Page Rank") +
  guides(color = "none") +
  theme_blank()
```


Karl
Craster
Cressen
Shireen
Orell
Salladhor
Alliser
Eddison
Davos
Dalla
Grenn
Samwell
Melisandre
Janos
Jojen
Val
Lancel
Jon
Meera
Aemon
Hodor
Nan
Theon
Illyrio
Stannis
Bran
Luwin
Belwas
Myrcella
Rickon
Drogo
Barristan
Jorah
Ramsay
Irri
Olenna
Daario
Rhaegar
Rickard
Missandei
Viserys
Lothar
Rakharo
Roslin
Daenerys
Ellaria
Worm
Gendry
Chataya
Aegon
Hoster
Kraznys
Walton
Doran
Anguy
Amory
Qyburn
Page Rank
0.01
0.02
0.03
0.04