Graph Networks Activity

Libraries Used

```
library(igraph)
library(tidyverse)
library(threejs)
library(readr)
library(knitr)
```

The Data

We have provided both a custom data set and an .RDS file with a list outlining various cases of dialogue from the tv show 'Friends', outlining instances of speech, the speaker, and the recipient of the speech.

```
node_list <- tibble(id = 0:5)
edge_list <- tibble(Root = c(0, 0, 0, 1, 1, 2, 2, 2, 3, 3, 4, 4, 5, 5, 5),
                    Destination = c(1, 2, 4, 2, 5, 0, 1, 5, 2, 4, 2, 3, 1, 2, 4))
```

```
friends_edgelist <- readRDS("edgelist.RDS")
```

Then, let's view the first 10 rows of each data set:

```
kable(friends_edgelist %>% head(10))
```

from	to	weight
a Casino Boss	a Tourist	1
a Casino Boss	Chandler	1
a Casino Boss	Joey	1
a Casino Boss	Phoebe	1
a Casino Boss	Rachel	1
a Crew Member	Alex	1
a Crew Member	Chandler	1
a Crew Member	Joey	1
a Crew Member	The Director	1
a Disembodied Voice	Phoebe	1

```
kable(edge_list %>% head(10))
```

Root	Destination
0	1
0	2
0	4
1	2
1	5
2	0
2	1
2	5
3	2
3	4

Since the Friends network is quite a bit larger than our custom data set, we'll use the filter command to filter out any occurances that don't involve the show's main characters.

```
friends <- c("Phoebe", "Monica", "Rachel", "Joey", "Ross", "Chandler")
edgelist_without <- friends_edgelist %>%
  dplyr::filter(!(from %in% friends & to %in% friends))
```

Lastly, to interact with the data most efficiently, we'll format each dataset as a graph network; notice, each employs a different method of doing so but arrives at the same result.

Custom data:

```
graph <- graph.data.frame(edge_list, directed = TRUE)
```

Friends data:

```
edgelist_matrix <- as.matrix(edgelist_without[ ,c("from", "to")])
friends_graph <- igraph::graph_from_edgelist(edgelist_matrix, directed = FALSE) %>%
  igraph::set.edge.attribute("weight", value = edgelist_without$weight)
```

Elements of a Graph Network

There are two main characteristics of a graph netowrk: the nodes, which are the individual items that form the netowrk, and the edges, which signify the connections between nodes. You can use basic commands to see the number of edges and vertices involved in any graph network as well as other characteristics such as the diameter, which is the largest number of connections between any two nodes.

Analysis of the custom data:

```
E(graph) # Indicates 15 edges
```

```
## + 15/15 edges from 86e3c2b (vertex names):
##  [1] 0->1 0->2 0->4 1->2 1->5 2->0 2->1 2->5 3->2 3->4 4->2 4->3 5->1 5->2 5->4
```

```
V(graph) # Indicates 6 nodes
```

```
## + 6/6 vertices, named, from 86e3c2b:
## [1] 0 1 2 3 4 5
```

```
get_diameter(graph) # Indicates a diameter of four
```

```
## + 4/6 vertices, named, from 86e3c2b:
## [1] 1 5 4 3
```

Analysis of the Friends data:

In addition to checking these basic network characteristics, certain commands tell you data about specific points.

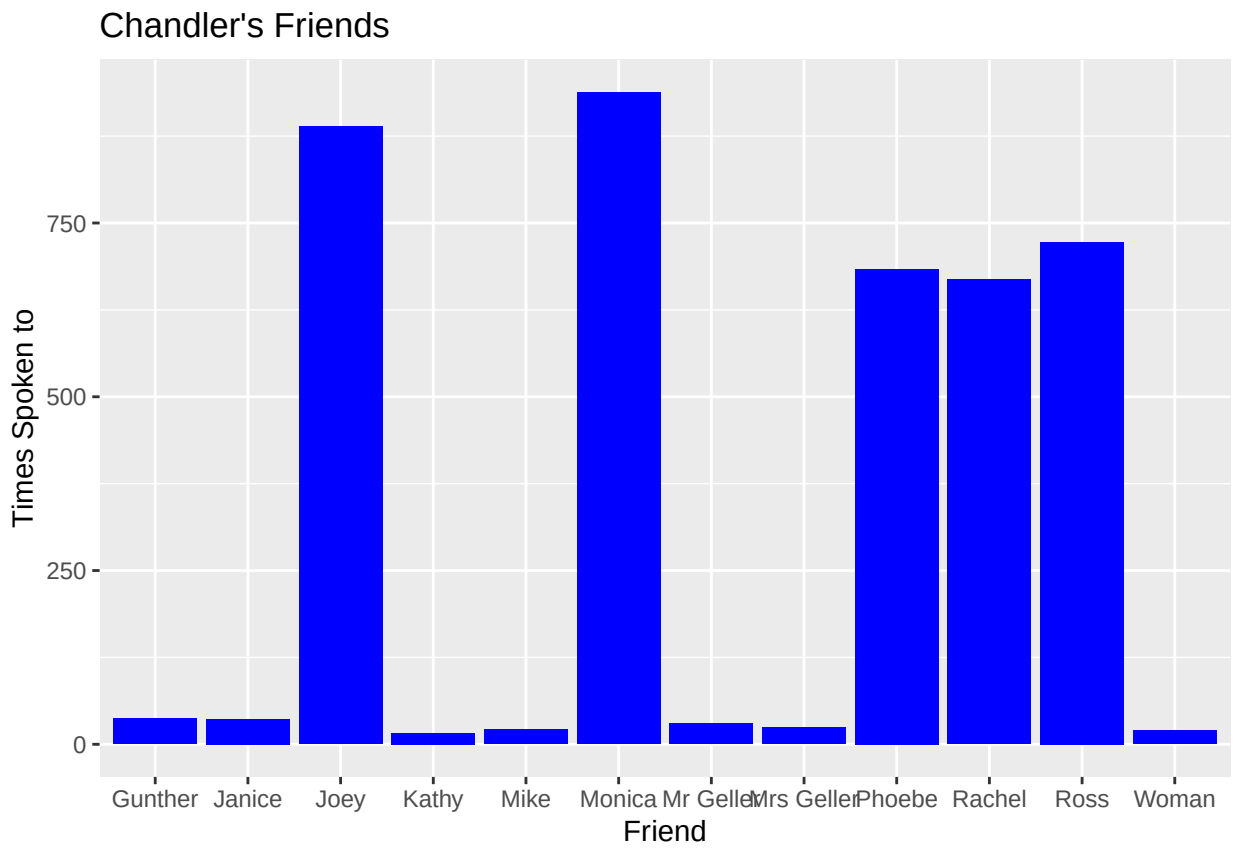
For example, the degree command can be used to tell us that Joey interacted with 348 characters:

```
degree(friends_graph, v = 'Joey')
```

```
## Joey
##  348
```

You can also use this data to create visualizations of characters' social patterns. For example, the histogram below shows the number of times that Chandler talked to each of his "friends", which, in this case, signifies anyone that he spoke to more than fifteen times.

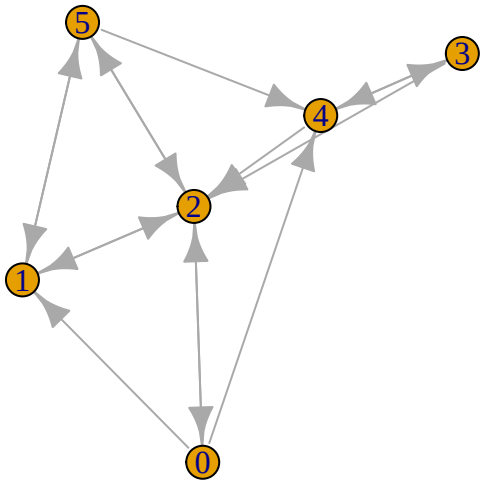
```
chandlers_friends <- filter(friends_edgelist, from == 'Chandler') %>%
  filter(weight > 15) %>% arrange(desc(weight))

ggplot(chandlers_friends, aes(x = to, y = weight )) + geom_bar(stat = "identity", fill = "Blue") +
  ggtitle("Chandler's Friends") + xlab("Friend") + ylab("Times Spoken to") +
  scale_fill_brewer(palette = "Blues")
```



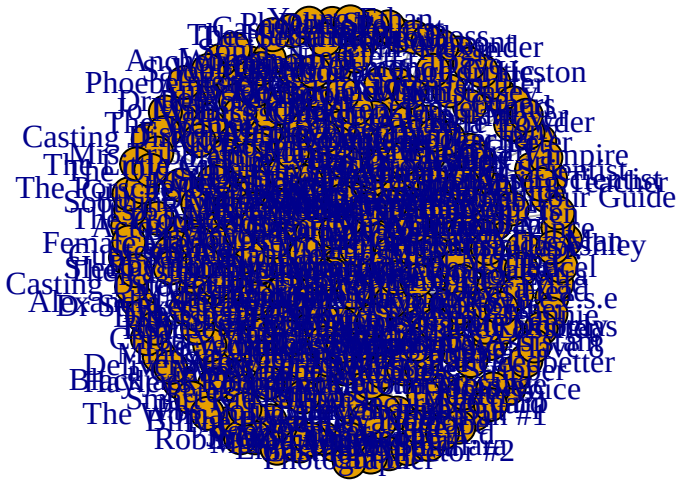
Visualizing the Network

Now that the basic elements of a graph network are understood, it's time to visualize them. One way to do this is by making an ego graph, which centers around a single node, the ego.

```
graph_ego <- make_ego_graph(graph, 3, '0', mode = c('all'))[[1]]
fr_graph_ego <- make_ego_graph(friends_graph, 1, 'Joey', mode = c('all'))[[1]]
plot(graph_ego)
```

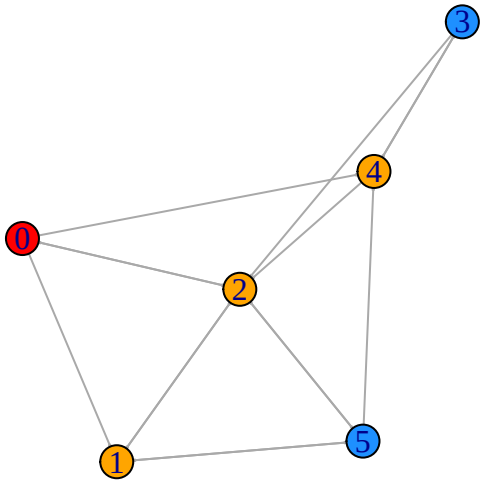



```
plot(fr_graph_ego)
```



As you can see, this still produces results that are not particularly helpful, especially for the monstrous Friends netowork. At the very least, we can add some colors to the homemade data that indicate each node's proximity to the ego, number zero (the Friends network will take a little more).

```
colors <- c("red", "orange","dodgerblue", "cyan")
dists <- distances(graph_ego, v = '0')
V(graph_ego)$color <- colors[dists+1]
plot(graph_ego, edge.arrow.size = .05)
```



For the Friends graph, we'll use the louvain_cluster command to organize the network into distinct communities, and assign those communities to the graph object to encourage additional organization. Then, we'll create subgraphs out of each community so they can be analyzed independently and sorted using dplyr commands for analyzing the nodes with the most betweenness, or involvement within the network. The resulting list will show each community's member count and its most central member.

```
# run louvain with edge weights
louvain_partition <- igraph::cluster_louvain(friends_graph, weights = E(friends_graph)$weight)
# assign communities to graph
friends_graph$community <- louvain_partition$membership
# see how many communities there are
unique(friends_graph$community)
```

```
## [1] 4 6 1 7 3 8 5 2
```

```
communities <- data.frame()
for (i in unique(friends_graph$community)) {
  # create subgraphs for each community
  subgraph <- induced_subgraph(friends_graph, v = which(friends_graph$community == i))
  # get size of each subgraph
  size <- igraph::gorder(subgraph)
  # get betweenness centrality
  btwn <- igraph::betweenness(subgraph)
  communities <- communities %>%
    dplyr::bind_rows(data.frame(
      community = i,
```

```
      n_characters = size,
      most_important = names(which(btwn == max(btwn)))
    )
    )
}
knitr::kable(
  communities %>%
    dplyr::select(community, n_characters, most_important))
```

community	n_characters	most_important
4	128	Chandler
6	109	Joey
1	105	Phoebe
7	93	Rachel
3	105	Monica
8	94	Ross
5	8	Intercom
2	8	Bandleader
2	8	Dennis Phillips

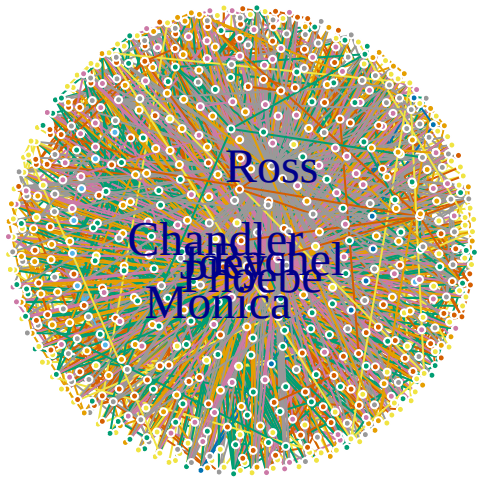
Since three of these groups are notably smaller and led by insignificant characters, we'll create subgraphs of each group and filter them out by group size to ensure that only the groups lead by main characters are included.

```
top_five <- data.frame()
for (i in unique(friends_graph$community)) {
  # create subgraphs for each community
  subgraph <- induced_subgraph(friends_graph, v = which(friends_graph$community == i))
  # for larger communities
  if (igraph::gorder(subgraph) > 20) {
    # get degree
    degree <- igraph::degree(subgraph)
  }
}
```

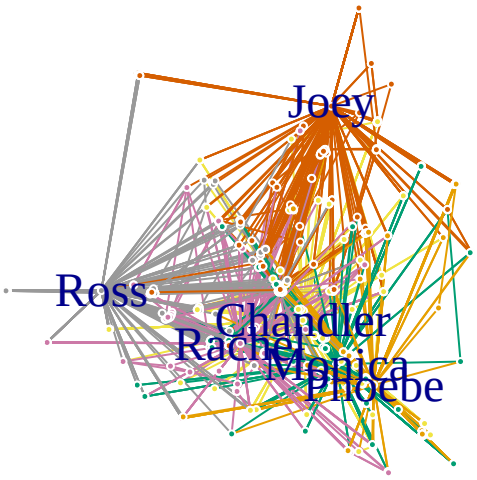
FINALLY, it is time to visualize the network in a way that is visually pleasing and reasonably organized. After adding some colors to represent a node's community, scaling the nodes by degree, and adding colored edges to show the root of the edge (or speaker of the dialogue), it's time to plot the graphs. Layout one, the spherical layout, is more visually pleasing but layout two is clearly preferable as it clusters groups together and bases node placement on community involvement and betweenness.

```
# Scaling by degree and coloring by community
V(friends_graph)$size <- 3
V(friends_graph)$frame.color <- "white"
V(friends_graph)$color <- friends_graph$community
V(friends_graph)$label <- V(friends_graph)$name
V(friends_graph)$label.cex <- 1.5
# Coloring by speaker
edge.start <- ends(friends_graph, es = E(friends_graph), names = F)[,1]
E(friends_graph)$color <- V(friends_graph)$color[edge.start]
E(friends_graph)$arrow.mode <- 0 # only label central characters
```

```
v_labels <- which(V(friends_graph)$name %in% friends)
for (i in 1:length(V(friends_graph))) {
  if (!(i %in% v_labels)) { V(friends_graph)$label[i] <- "" }
}

l1 <- layout_on_sphere(friends_graph)
plot(friends_graph, rescale = T, layout = l1, main = "'Friends' Network - All Seasons")
```




```
l2 <- layout_with_mds(friends_graph)
plot(friends_graph, rescale = T, layout = l2, main = "'Friends' Network - All Seasons")
```



Acknowledgements

Special thanks to Keith McNulty's article "Community Detection in R Using Communitues of Friends Characters" for adding some popular culture to this activity.