PSTAT 194CS Final Project Write-Up, Group 5, Wikipedia Animal Articles

Lucas Webster, Will Mahnke, Sam Caruthers

June 07, 2024

Package Loading

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

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

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

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:igraph':
##
##     as_data_frame, groups, union

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

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

Introduction

The dataset we selected features nodes as Wikipedia articles and edges as the hyperlinks between them as of December 2018. The dataset involved two csv files and one json file for each of the animals being a Chameleon, Crocodile, and Squirrel. The csv files contained the edgelist and target data, and the JSON file contained the informative nouns of each node. Our goal was to find a correlation between the amount of shared informative nouns (or features) between two nodes, and whether or not that correlated to a shared edge. We also hoped to group the different nodes by the ones that had the highest target, or monthly site visits, in clicks, to see if there were any patterns or correaltions corresponding to target value. This graph is undirected, as specified by the read.me file in the data, and is unweighted. The dataset came with three sets of data, one for crocodiles, one for chameleons, and one for squirrels. Each dataset was indexed independantly, meaning the data could not be meaningfully combined. We decided to each take on an animal and its dataset, perform analysis, and compare and contrast the animals after analysis was conducted.

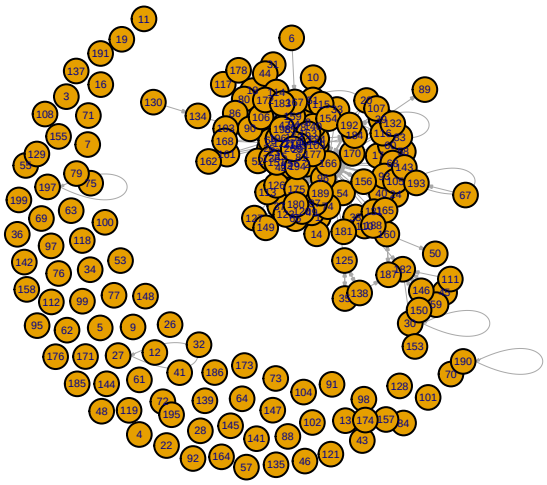
Methodology

Our first challenge stemmed from reading the JSON file into our networks. Which was taxing from a mental and memory standpoint. Once we created our subnetworks, after a few visualizations we learned that we had to remove all nodes that did not have any edges (no hyperlinks on the page), or nodes whose hyperlinks were only pointing to websites in the network but not the selected subgraph (self-loops). Our implementation of this was thanks to the use of base R, as well as the igraph, igraphdata, Matrix, rjson, dplyr, and ggplot2 packages.

Analysis

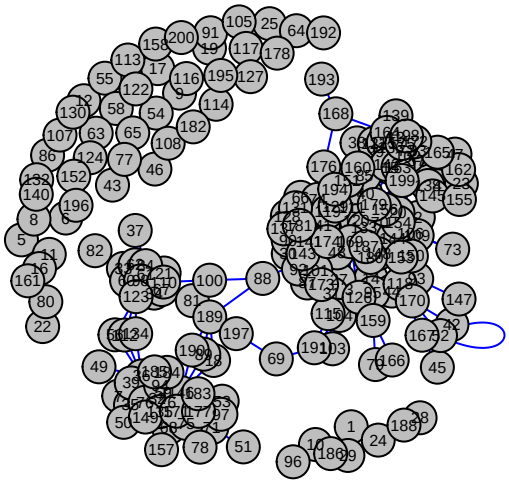
Original Graph

Lucas: Squirrel



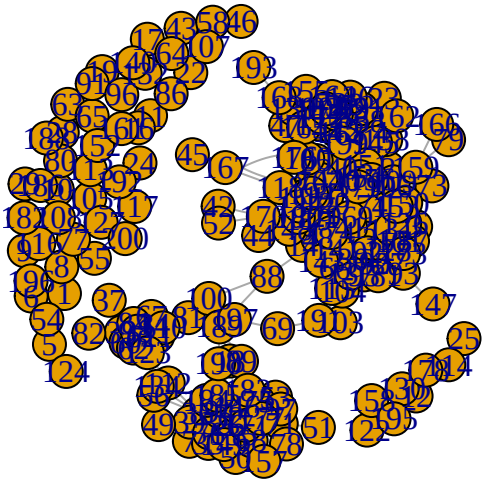
As referenced above, The original plot of the subgraph has a number of issues that would inhibit analysis. First off, there are many nodes with no edges. This will drive down nearly all metrics, and not tell us anything about edge likelihood when associated with noun count. Additionally, there are quite a few nodes with edges to themselves, or edges which connect to nodes that are not in the sampled subgraph. This gives us faulty data. I proceeded by filtering out nodes which did not meet a degree requirement (0,1, and 2) until I reached a number of nodes that seemed high enough, and did not have many self loops or nodes who's edges left the subgraph. I then simplified this subgraph to remove self-loops, as they tell us nothing about page-to-page travel. In the end there were 79 nodes and 441 edges, from an original subnetwork of 200 nodes and 585 edges. This uses the auto-layout.

Will: Crocodile



A "basic" plot for the sub-network looks incredibly clunky. It's hard to see any edges because the vertices are too big which makes it hard to get see any relationships between the nodes. The metrics we found later also told us that a significant proportion of the nodes have degree 0. To make the visualization easier, we removed the nodes with degree zero and organized the nodes in a ring using `layout = layout_in_circle` . We also removed the loops for particular nodes for a cleaner visualization.

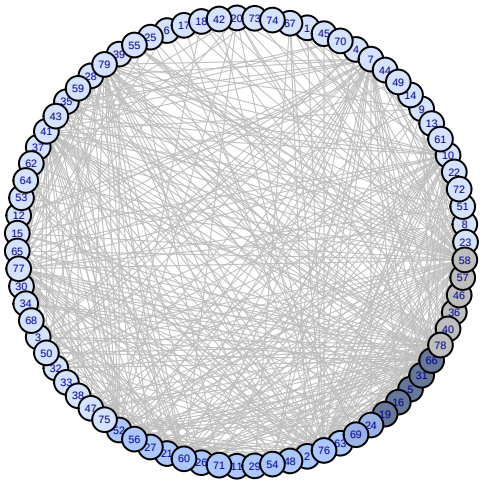
Sam: Chameleon



This basic plot is clearly not a great visualization of this sub-network. There are several nodes that are not attached to anything, and some nodes that have loops to themselves, which we have no way of interpreting.

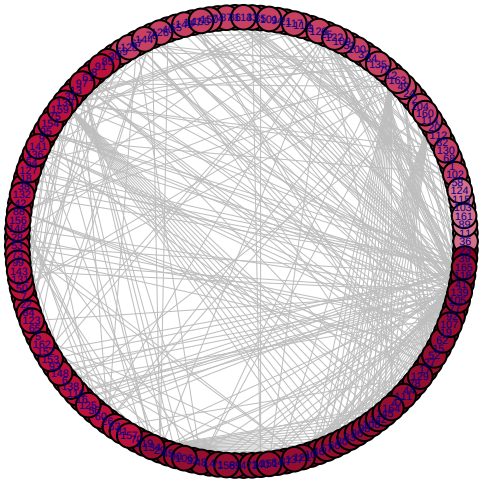
Revised Graph: Target Count

Lucas: Squirrel



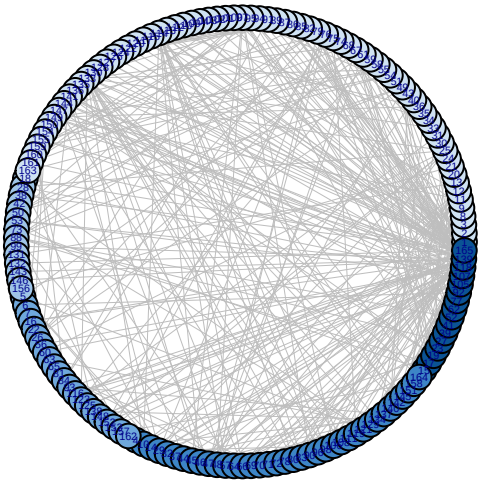
This plot organizes the nodes by their target value, into seperations of 0-24999, 25000-49999, 50000-74999, 75000-99,999, 100,000+ by color, with darker blues associating with higher target values, and gray associating with a target value of 100000 or greater. The plot organizes the nodes by these colors, with the lightest color starting at "3:00", and moving darker counter-clockwise. We can observe that there are more and more edges the farther around the circle you travel, with the highest concentration of edges seeming to stem from the dark blue area. This shows that edges seem to concentrate around nodes with higher target values, meaning that pages with more monthly traffic contain more hyperlinks to other pages. This uses the circle layout.

Will: Crocodile



Taking out the nodes with degree zero left only 75. The circular layout shows there are a few nodes that are much more popular than most, which we analyze further in community detection and network metrics. The final step to creating a meaningful visualization is incorporating our target data and noun list. To utilize this data, we'll uses different shades of a color to distinguish the target data between nodes and use edges to show which nodes share a particular amount of nouns.

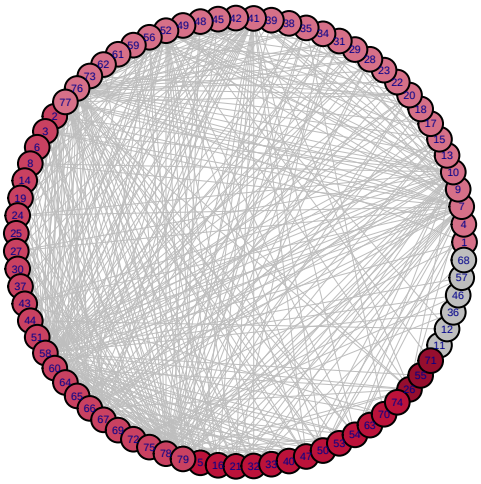
Sam: Chameleon



Target subgraph which is darker as the target value of the node increases. You can see that about half the graph has the lightest color, which means they have a target value below 2,000. THen it scales up to the dark blue which represents nodes with a value greater than 20,000.

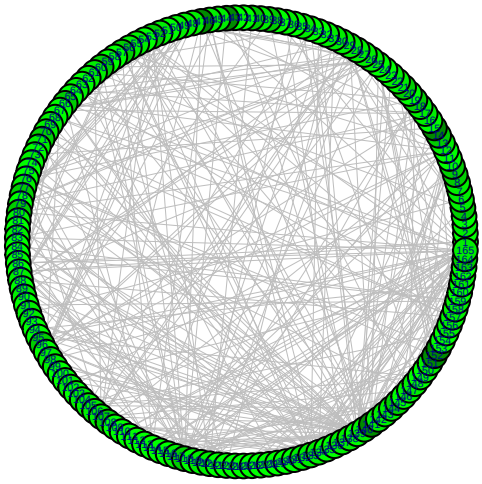
Revised Graph: Noun Count

Lucas: Squirrel



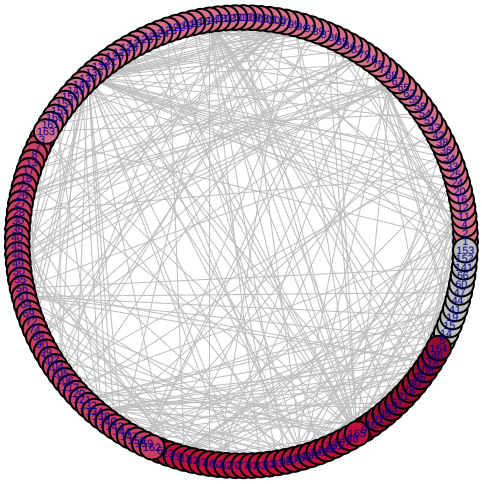
This plot was colored and sorted in the same fashion, but for the color red, and for the informative noun count in each article. The intervals were 0-14, 15-24, 25-34, 35-44, 45+. There seems to be the most edge clustering around 15-24 interval, meaning increasing noun count is not indicative of hyperlinks like target is. This graph is undirected, meaning that these pages may either have a bunch of hyperlinks leading to them, or a bunch of hyperlinks leading away from them. It also means nodes with high target values sit in a mid-noun range, seeming to mean target and nouns are not positively or negatively correlated at first glance. This also uses the circle layout.

Will: Crocodile



There doesn't seem a pattern in noun list length when it comes to the connection of points to the popular node, which logically has a long list. It also seems that the vertices in the filtered sub-graph have long noun lists.

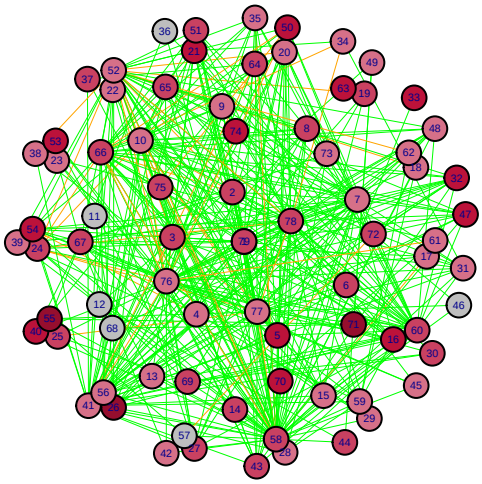
Sam: Chameleon



The degree ordered subgraph is more evenly distributed among the different categories. The lightest pink represents nodes with degrees less than 15, and this increment increases by 10 until the gray which represents nodes with a degree greater than 45. The visualization shows that over half the websites in the subnetwork have a degree of greater than 15.

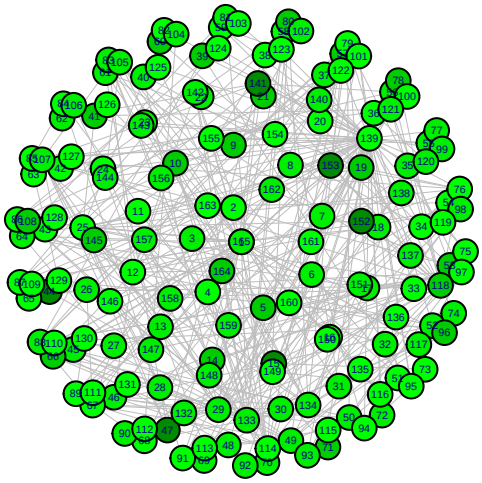
Revised Graph: Nouns Shared

Lucas: Squirrel



This plot colors nodes based on noun count as before, but also colors edges based on if the nodes they are connecting share two or more nouns. Green edges indivate 2+ nouns are shared, and orange edges indicate 0 or 1 noun is shared. About 90% of edges are green and about 10% of edges are orange. This indicates that there is a 90% chance that an edge will indicate that the nodes it connects share 2 or more nouns, and visa versa. When bumped to 3 or more, this number drops significantly to about 65%, and when lowered to 1 or more, it rises to about 99%. This puts 2 or more nouns as a reasonable assessment of how many nouns two connected vertices will share for this subnetwork.

Will: Crocodile



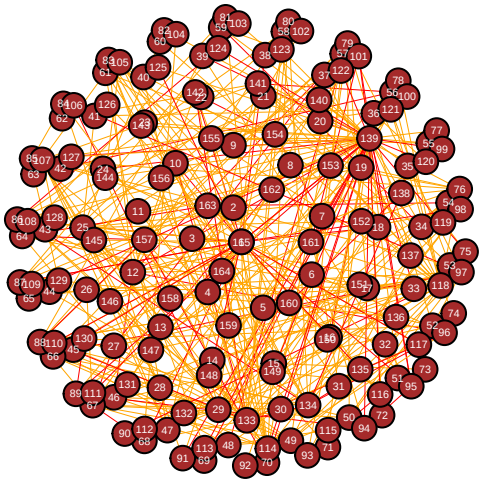
Most of the vertices connected together share more than two nouns. However, the presence of a connection where the nodes only share two words at most sparks intrigue.

Using the coloring of the edges, we can calculate the proportion of edges are between vertices that share more than two nouns, which shows that about 83% of connected nodes share more than two nouns.

Sam: Chameleon

```
##  [1] 1465 1503  929 1794 2297 2753  735 2539  415  598 1471
```

```
## [1] 177
```



This graph colors edges orange if they shared two or more keyword nouns between the two nodes, and red if they 0 or 1 shared nouns in common. Since roughly 80 percent of these edges share two or more nouns, we interpret that two websites having an edge suggests they most likely share two or more nouns.

Metrics

Lucas: Squirrel

```
##                                      Metrics
## Articluation Points                  "58, 68"
## AP Degree                            "63, 3"
## AP Target                            "126190, 20636"
## AP Nouns                             "20, 46"
## Diameter                             "5"
## Transitivity                         "0.43"
## Density                              "0.14"
## Most Nouns (70) Degree               "3"
## Highest Betweenness Centrality (58) "1279.83"
## Mean Betweenness Centrality          "39.68"
## Std Dev of Betweenness Centrality    "152.39"
## Mean Target                          "28926.09"
## Std Dev of Target                    "34886.71"
```

This subnetwork is not connected, but has two articulation points, node 58 and 68. This means these nodes hold a significant portion of the subnetwork's structure, and it's central cluster, as seen in the below communities, would fracture into multiple pieces should these nodes be removed from subnetwork. Interestingly, these articulation points, nodes 58 and 68, have a respective degree of 63 and 3. This makes node 58 highly connected, making its role as an articulation point obvious, while node 68 has only degree 3, meaning it likely is the only connection between some clusters of nodes which would otherwise be unconnected. Furthermore, thes articulation points have target values of 126190 and 20636 respectively. This explains node 58's high degree, as Wikipedia pages with high traffic are likely to also be highly editted, and therefore have more hyperlinks. Their respective noun count is 20 and 46, once again showing that noun count is not an indicator of degree or traffic. The network has a diameter of 5, meaning if you were to start on a Wikipedia page in this subnetwork, and only travel to other pages in this subnetwork using hyperlinks, you could visit at most 5 networks. This subnetwork has a transitivity of 0.43, only about 8% higher than the actual network, meaning nodes of degree 0, 1, and 2 make up significantly less 'triangular communities' than nodes of degree two or higher. It can also be noted the node with the highest betweenness centrality is the articulation point node 58, with 1279, meaning it is crucial in connecting two pages along their shortest path, ie, if one were to do a Wikipedia page link speed run. The mean betweenness centrality is 39 and it has a standard devation of

152, meaning there are a few pages that are crucial to this speed run idea, and a bunch that have little to no bearing. The average target value was 28926 with a standard deviation of 34886, meaning a bunch of pages get little traffic, and there are a few heavy hitters.

Will: Crocodile

```
## .
##  1  2  3  4  5  6  7  8  9 10 11 12 13 15 16 22 23 37 55
## 50 29 20 16 17 10  5  1  1  3  1  2  2  3  1  1  1  1  1
```

```
## [1] 0.02653363
```

```
## [1] 0.1854571
```

The table for the degree of the nodes indicates that most of the nodes in the sub-network don't have any connections to other articles. The table shows about 63 percent of the nodes don't have any other connection and all but three nodes have less than three connections. The distribution of degree indicates that apart from some pairs in the network, one particular node is at the center of most of the group. There are a couple of connections outside of this main group but most of the nodes in the network aren't connected to others. The density of the sub-network reflects how sparse the connections are. Most articles are related to or come from that most popular node and have very little to do with each other, which is supported by the incredibly low transitivity.

In the context of the network, the density and degree imply that a couple of pages out of the sub-network are the most important/general pages. Since the nodes corresponding to pages are anonymous, we have to infer that these popular pages are broad and serve as an outline for more specific topics, the nodes that are connected to that popular node. Nodes with no connections are likely articles that would come at the end of a long rabbit hole, nuanced material that's unrelated to anything else.

```
## [1] 4.173508
```

```
## [1] 9
```

The average path length of the network is about two, which means that most of those paths go through just the popular node. The diameter being three reflects this much, but also indicates there are a few paths between nodes that have one extra node along with the most popular node.

The low average path length and diameter reflect the conclusions drawn from the networks density and nodes' degrees. The popular page serves as a connection between a bunch of other pages. The diameter being three implies that going between the two "farthest" pages requires one page on top of connecting through the main page.

```
## + 16/165 vertices, from 59e6320:
##  [1] 139 165 137  93  26  56 141 107  29  67  75 129 140 138 154  13
```

```
## [1] 165
```

```
## [1] 8052.517
```

```
## [1] 835.6065
```

The filtered network (temp) has six articulation points. Most of these can be traced back to the nodes that have the most connections, which is logically because they're the node holding a lot of others together. These articulation points represent crucial webpages that are likely important to understanding a sub-topic of crocodiles.

The large max betweenness centrality reflects ideas introduced previously about one popular node being the center for most of the filtered network. It acts as the middle man between a lot of vertice combinations so it would make sense for it to have that high betweenness centrality. The high standard deviation of betwenness centrality for the network indicates the most populat node (along with the few other main connectors of

the network) have high betweenness centrality while the bulk of the nodes are going to a low betweenness centrality.

```
## [1] 10186.6
```

```
## [1] 22681.47
```

The average traffic for the nodes in the network is about 8800, which indicates that the network as a whole is getting a fair amount of traffic during the studied period of time. However, with a standard deviation of about 32000, there are a lot of nodes that are likely outliers in the amount of traffic they get (which is true considering the highest traffic is about 210000).

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

```
## $membership
##   [1] 1 1 1 2 1 3 4 5 6 1 1 1 5 1 7 1 1 1 1 1 8 4 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 7 1
##  [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 6 1 9 1 1 1 1 1 1 1
## [112] 9 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 5 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [149] 1 1 1 1 1 4 1 8 1 1 1 1 1 2 1 1 1
##
## $csize
## [1] 147   2   2   3   3   2   2   2   2
##
## $no
## [1] 9
```

The results from investigating the density of the network imply the network isn't connected, and calling `components` reveals the big group of nodes connected to that most popular node with a bunch of smaller components of either the less popular nodes, the pairs, or the bulk of nodes (ones with degree 0) that are isolated.

Sam: Chameleon

```
## Warning: `articulation.points()` was deprecated in igraph 2.0.0.
## i Please use `articulation_points()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
## + 16/165 vertices, from 2b11d37:
##  [1] 139 165 137  93  26  56 141 107  29  67  75 129 140 138 154  13
```

```
## [1] 10.625
```

There were 17 articulation points in my 131 node subgraph. These articulation points correlate to broad and longer Wikipedia articles that have more specific articles linked to them in their respective communities. This is why the Articulation point mean degree is 8.65 which is much larger than the average mean degree which is a little under 2.

```
## [1] 0.3136243
```

```
## [1] 0.1854571
```

The transitivity of the entire network vs the subgraph serves as a check that the sample we took is reflective of the general network. Since $0.31 \sim 0.30$, we know we have a good sample.

The node with the most nouns reflects the webpage with the most keyword nouns, and in the case of my dataset the most nouns was 78. I anticipated the node with the most nouns (NWMN) to have a lot of edges because edges do seem to be relatively correlated to nouns. However, it has a degree of 1 and a betweenness centrality of 0, which suggests that it is a relatively unimportant node in the network. Which means that

particular website might have a lot of niche words that classify as keywords because we do not exactly know how this dataset determines that attribute.

```
## [1] 8052.517
```

```
## [1] 206.6242
```

```
## [1] 835.6065
```

The highest Betweenness centrality in my subnetwork is 1031.225 which is quite large. This implies that the node has several edges, and several of those edges have a high betweenness themselves. The average betweenness centrality is 29 and the standard deviation is 117, which infers that the graph is very inconsistent with some nodes having a very high betweenness centrality, and other nodes not being frequented very often. This suggests that there is a lot of traffic between some wikipedia sites and their hyperlinks, while others are rarely accessed.

```
sum(targetList)/length(targetList)
```

```
## [1] 10186.6
```

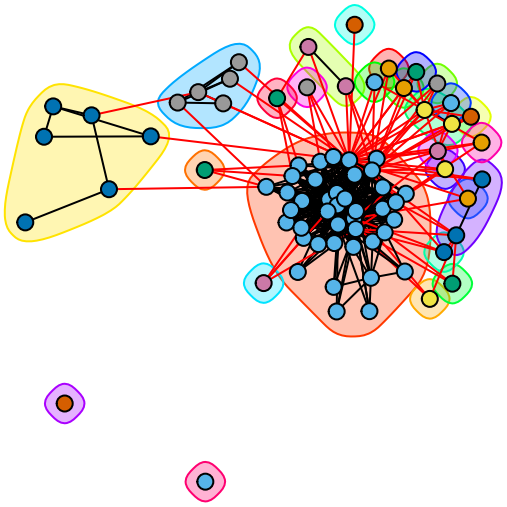
```
sqrt(var(targetList))
```

```
## [1] 22681.47
```

The average and standard deviation of the target attribute allows us to conclude similar information. The target attribute reflects how much traffic a particular node/website gets in a month, so an average of 9814 with a standard deviation of 28199 suggests again that some websites are a lot more popular than the average and others are barely visited at all.

Community Detection: Edge Betweenness

Lucas: Squirrel



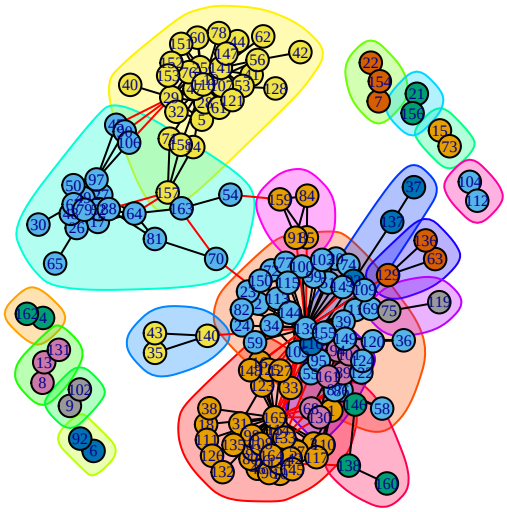
We constructed plots based on edge-betweenness to examine this speed-run philosophy mentioned above. A large cluster which contains node 58 lies in the center, and it has a lot of edges with leave its community to other communites. This means if one were to do a speedrun across communities, they would likely need to travel through this central community first. It has a modularity of 0.085, and has 27 communities, many with just one node, likely due to how many have such low betweenness centrality. It has 85 edges which cross communities total.

Will: Crocodile

```
## [1] 19
```

```
## Community sizes
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
## 34 36  2 31  2  3  3  2  2 21  2  3  4  3  6  2  4  2  3
```

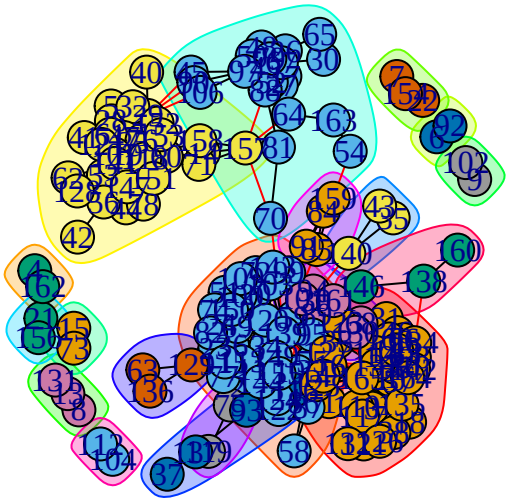
```
## [1] 0.6638954
```



`cluster_edge_betweenness` produces 13 different communities, all but three having only three members or less. A modularity of 0.602 means the network is relatively strong, but also indicates there could be some articulation points holding those communities together, namely the more popular nodes found when exploring the network metrics.

The plot of the communities reflect the analysis from the modularity. Every community is held together by a most popular node, with the plot only having one inter-community edge. This supports the claim from the network metrics section that all of nodes connected to their respective popular node are likely subsections of that popular node. The one inter-community edge is an outlier, so I decided to investigate how many nouns the nodes share when using `cluster_fast_greedy`.

Sam: Chameleon

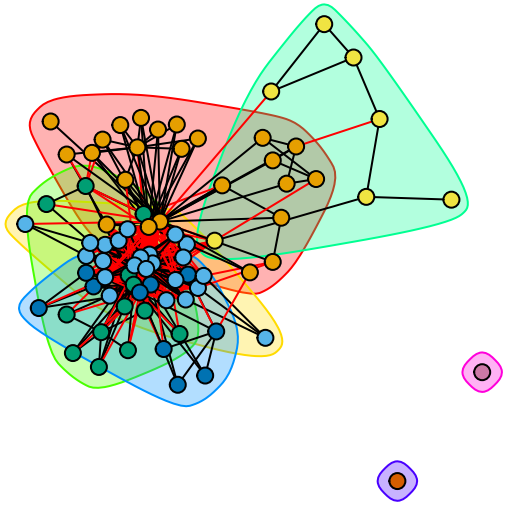


Network features one or two very large communities while the other clusters are more sparse. There are 11

intercommunity links and this network has a modularity of ~0.7 (0.733 from the edge between algorithm). This high modularity reflects the graph which shows a lot of connections between nodes in the bigger communities, with not too many connections between other communities. In the context of wikipedia articles this suggests that these large communities likely have a shared characteristic between each other which leads to several hyperlinks between them.

Community Detection: Fast Greedy

Lucas: Squirrel



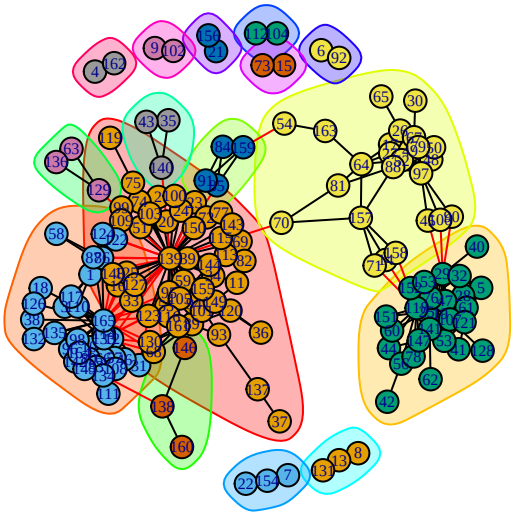
This fast greedy has less communities involved, but also less nodes in a singular community. This is because fast greedy maximizes modularity, and by grouping nodes into larger, easily separable communities, it can maximize this modulalrity. It has 285 edges which cross communities, which doesn't make as much sense, as I would have figured to maximize modularity, it would have kept this figure low in order make the communities easily separable. This algorithm has a modularity of 0.18 for this subnetwork.

Will: Crocodile

```
## [1] 16
```

```
## Community sizes
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
## 49 33 27 25  4  3  3  3  3  3  2  2  2  2  2  2
```

```
## [1] 0.6718368
```



It appears that using `cluster_fast_greedy` gave the same communities as `cluster_edge_betweenness`, which speaks to the importance of particular popular nodes in the network. But we'll investigate the nouns shared by the inter-community edge.

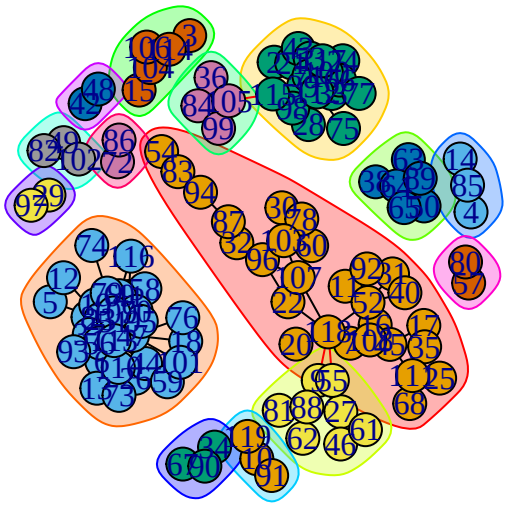
```
##  [1]   6  35  41  45  95  98  99 106 110 163 165 195 197 200 202 231 235 239 249
## [20] 261 281 298 305 314 316 317 319 321 324 326 327 329 337 344 347 353 356
```

The inter-community edge is edge 48. Using the edge number we can see how many nouns overlap.

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

The nodes (i.e. the webpages) between communities only share two words in common. Considering the length of some nodes' noun lists are much larget than that, we'll also investigate the proportion of vertices connected when they share a particular amount of nouns.

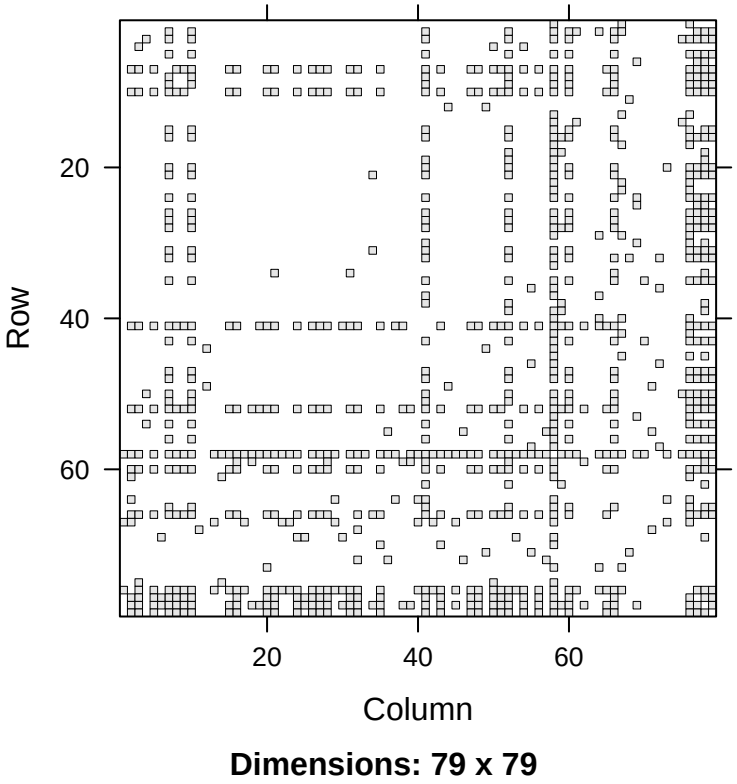
Sam: Chameleon



The cluster fast greedy algorithm generates a very similar result. Here we have a modularity of 0.728, and a very similar plot is generated.

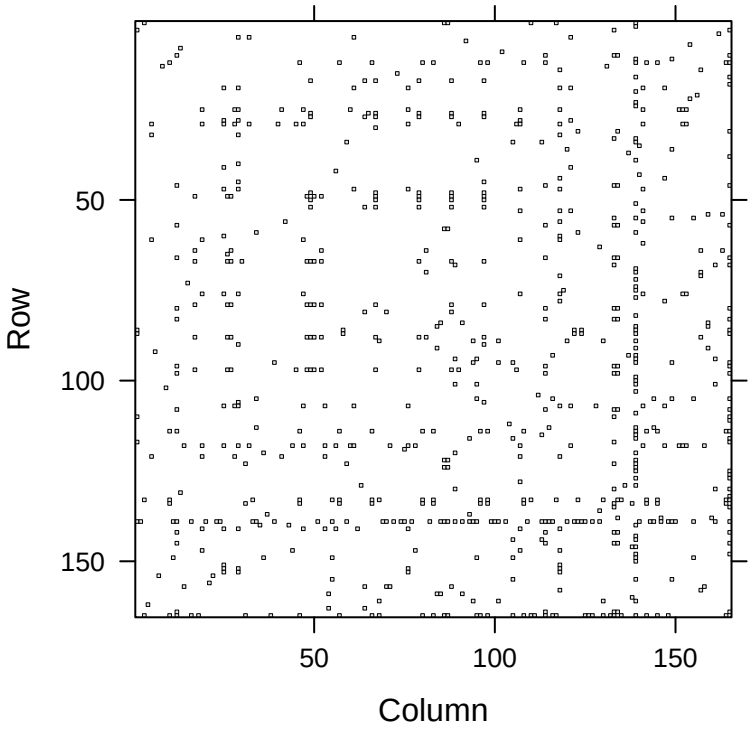
Adjacency Matrix: Original

Lucas: Squirrel



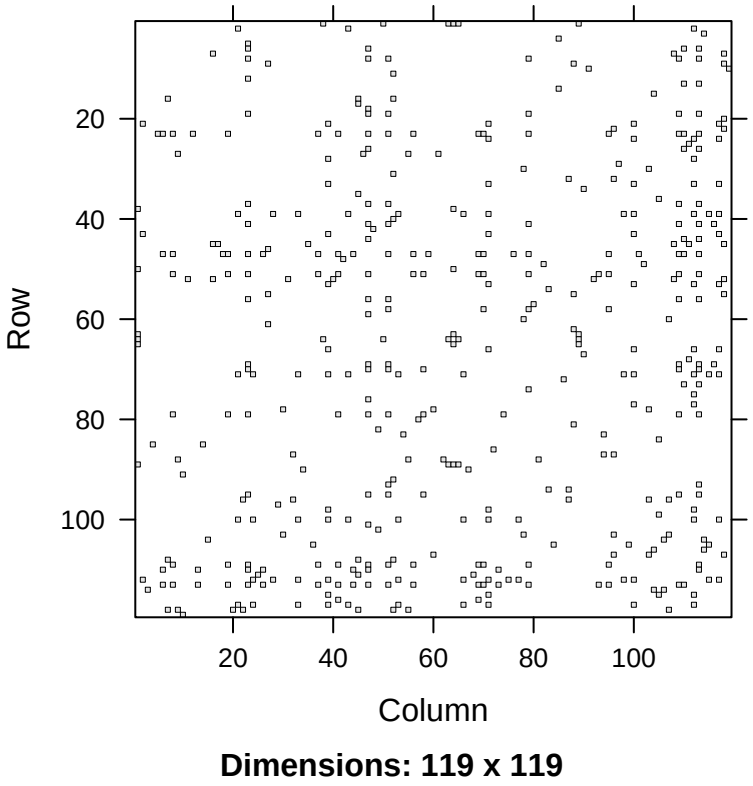
This is the original adjacency matrix. There are no obvious patterns.

Will: Crocodile



The basic adjacency matrix for the filtered network doesn't show any particular patterns, it's easier to see from the visual which nodes have the most connections, but we also experimented with reordering the nodes in the matrix based on communities, target data, and degree.

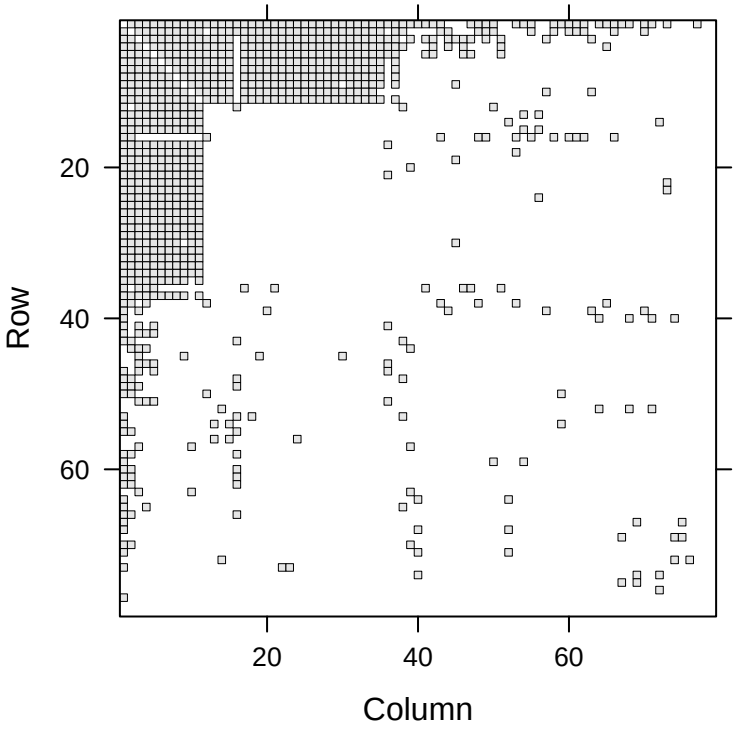
Sam: Chameleon



The original adjacency matrix does not have any particular patterns that allow us to better understand the data set, so we ordered the matrix by community and degree.

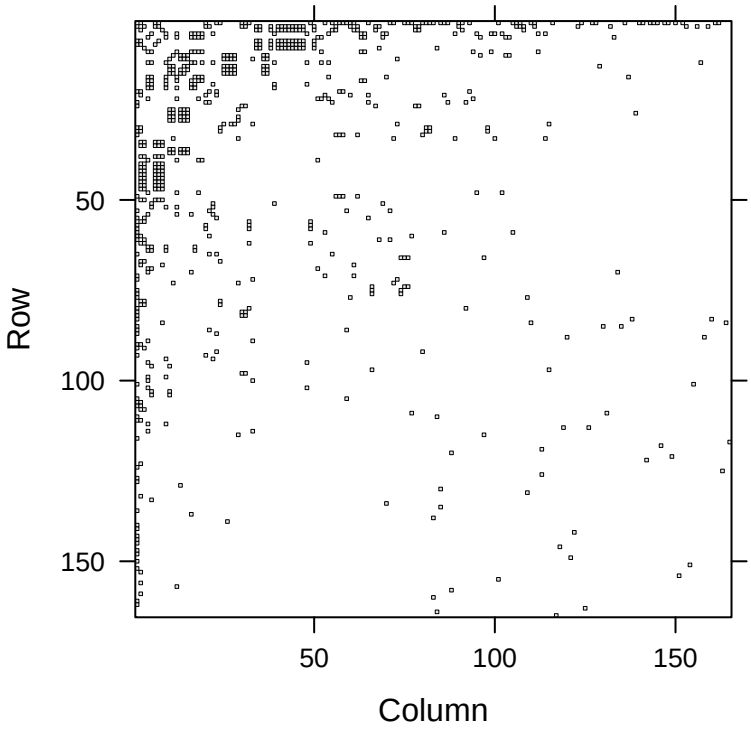
Adjacency Matrix: Degree

Lucas: Squirrel



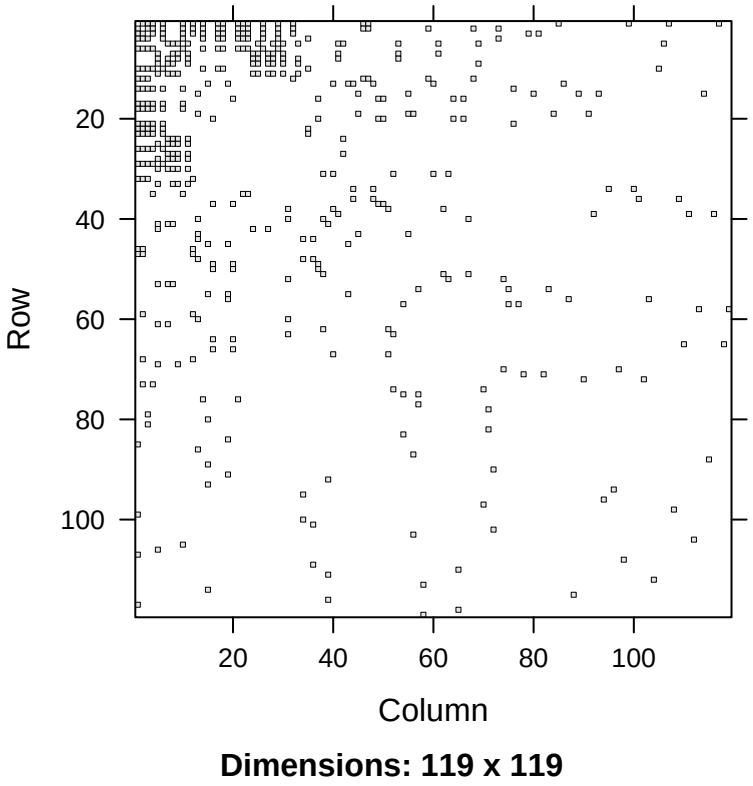
This adjacency matrix is sorted based on degree. Here we see a stark dropoff in degree about a quarter of the way through the matrix, and again about halfway through. It seems degree is stagnated; there are pages with a ton of links, a moderate amount, and almost none.

Will: Crocodile



Reordering the nodes based on degree doesn't share as much insight as the previous example. The visual shows all of the connections the most popular node has. The top left of the plot does show some slight denstiy between nodes with high degree, but the rest of the visual doesn't help as a bulk of the nodes have a degree of 1.

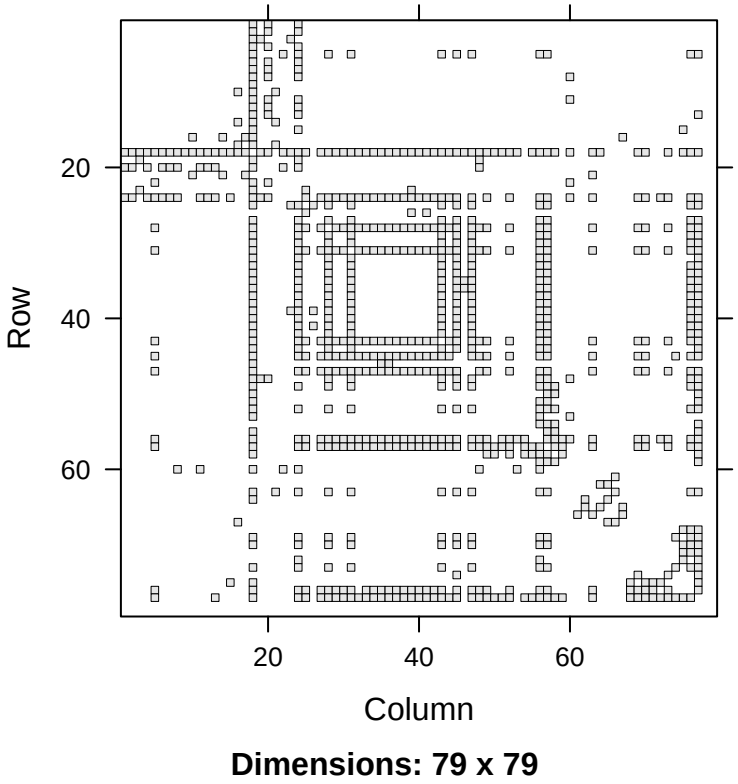
Sam: Chameleon



The ordering of the adjacency matrix by degree gives a predictable result, which is that there is a high concentration of nodes in the top right of the graph. This makes sense because the nodes with the most edges should have a lot of different connections up and down the rows and columns tied to the top corner, which gives the matrix that rotated 'L' shape.

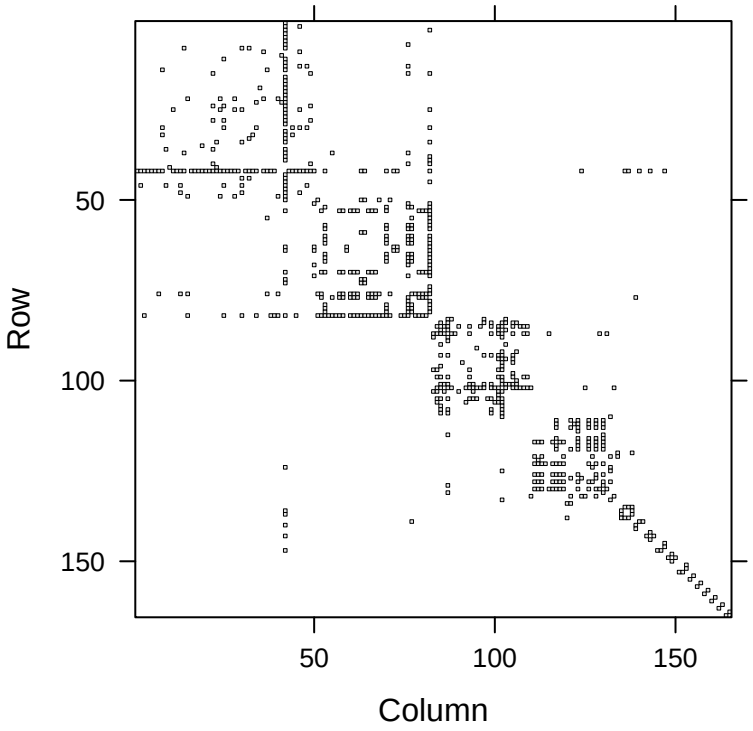
Adjacency Matrix: Fast Greedy

Lucas: Squirrel



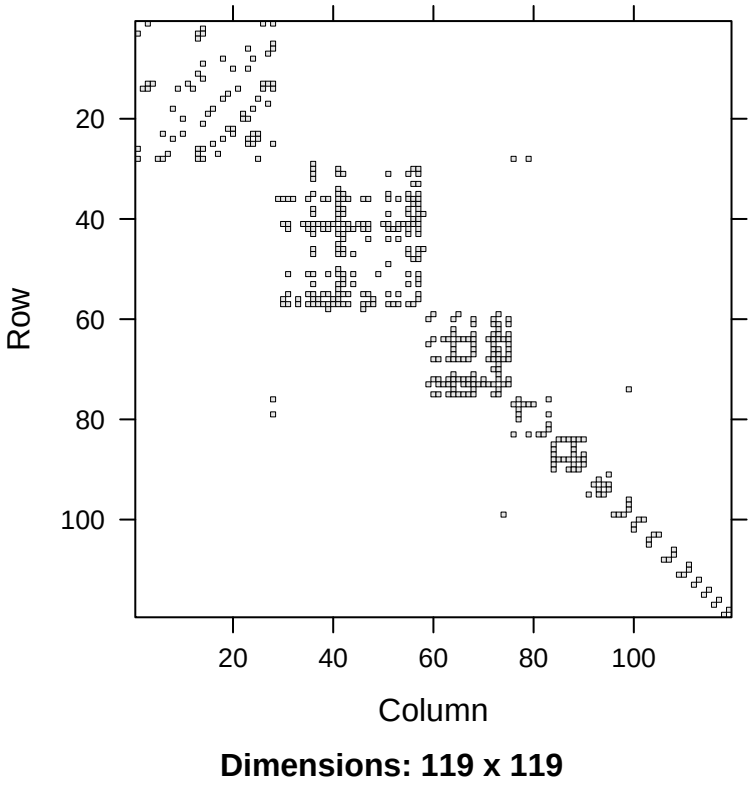
When re-ordered based on the communities from the fast_greedy community detection algorithm, we make out communities, and they seem to be centered on nodes which have high degrees. This would explain why there are so many cross community edges in the fast_greedy algorithm as mentioned above.

Will: Crocodile



The matrix organized for communities shows the communities centered around a particular popular node. The ordering of the communities also shows the drop off in size of the communities to the bottom right corner which is effectively isolated nodes.

Sam: Chameleon



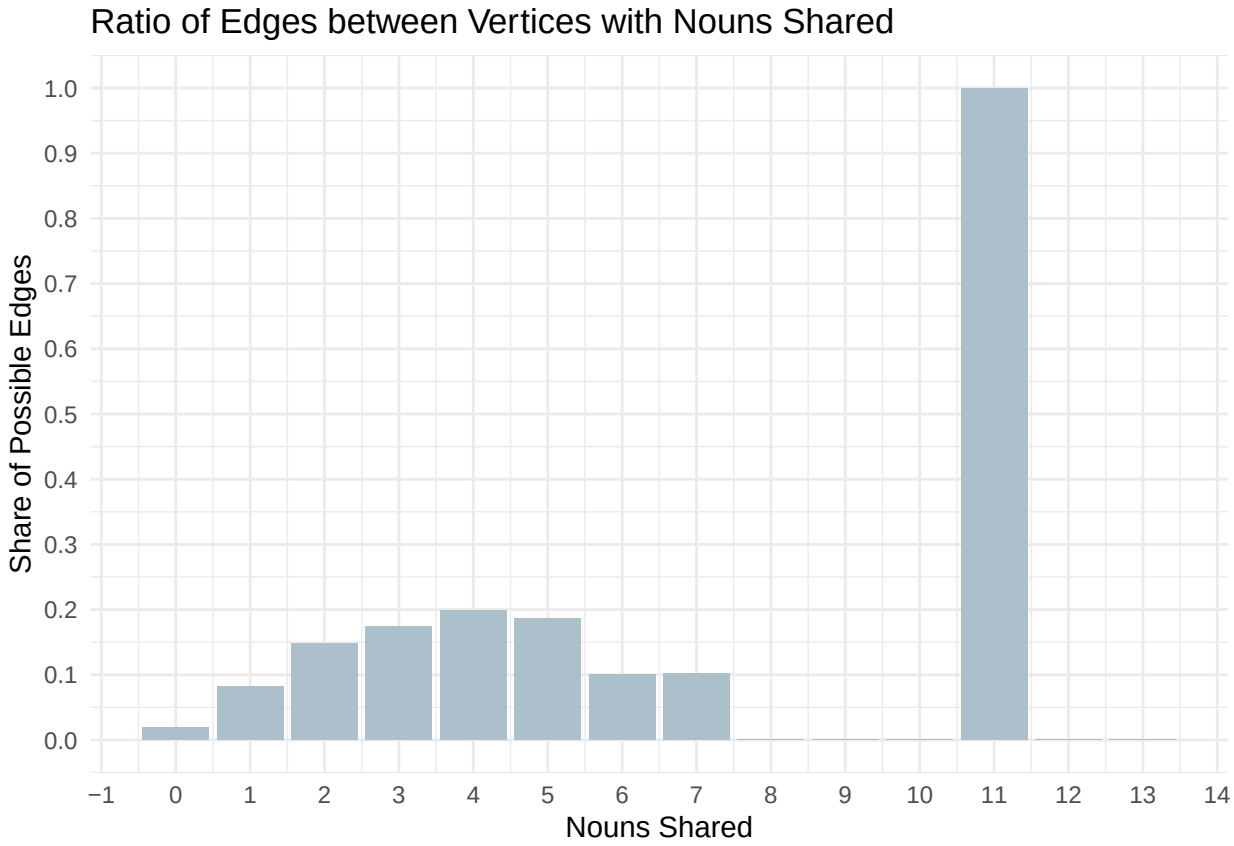
This first box in the top corner represents the biggest community seen earlier, followed by the next biggest square representing the smaller community and so on. The dots scattered reflect intercommunity links between nodes. This matrix clearly shows how these wikipedia articles are really only connected to their communities with sparse links that tie different nodes together.

Results

Nouns Table and Bar Graph

Lucas: Squirrel

```
##       Amount of Shared Nouns Vertice Combinations Num of Edges      ratio
##  [1,]                      0                  210           4 0.01904762
##  [2,]                      1                  503          41 0.08151093
##  [3,]                      2                  759         112 0.14756258
##  [4,]                      3                  735         128 0.17414966
##  [5,]                      4                  484          96 0.19834711
##  [6,]                      5                  236          44 0.18644068
##  [7,]                      6                  100          10 0.10000000
##  [8,]                      7                   39           4 0.10256410
##  [9,]                      8                    9           0 0.00000000
## [10,]                      9                    2           0 0.00000000
## [11,]                     10                    0           0 0.00000000
## [12,]                     11                    2           2 1.00000000
## [13,]                     12                    1           0 0.00000000
## [14,]                     13                    1           0 0.00000000
```



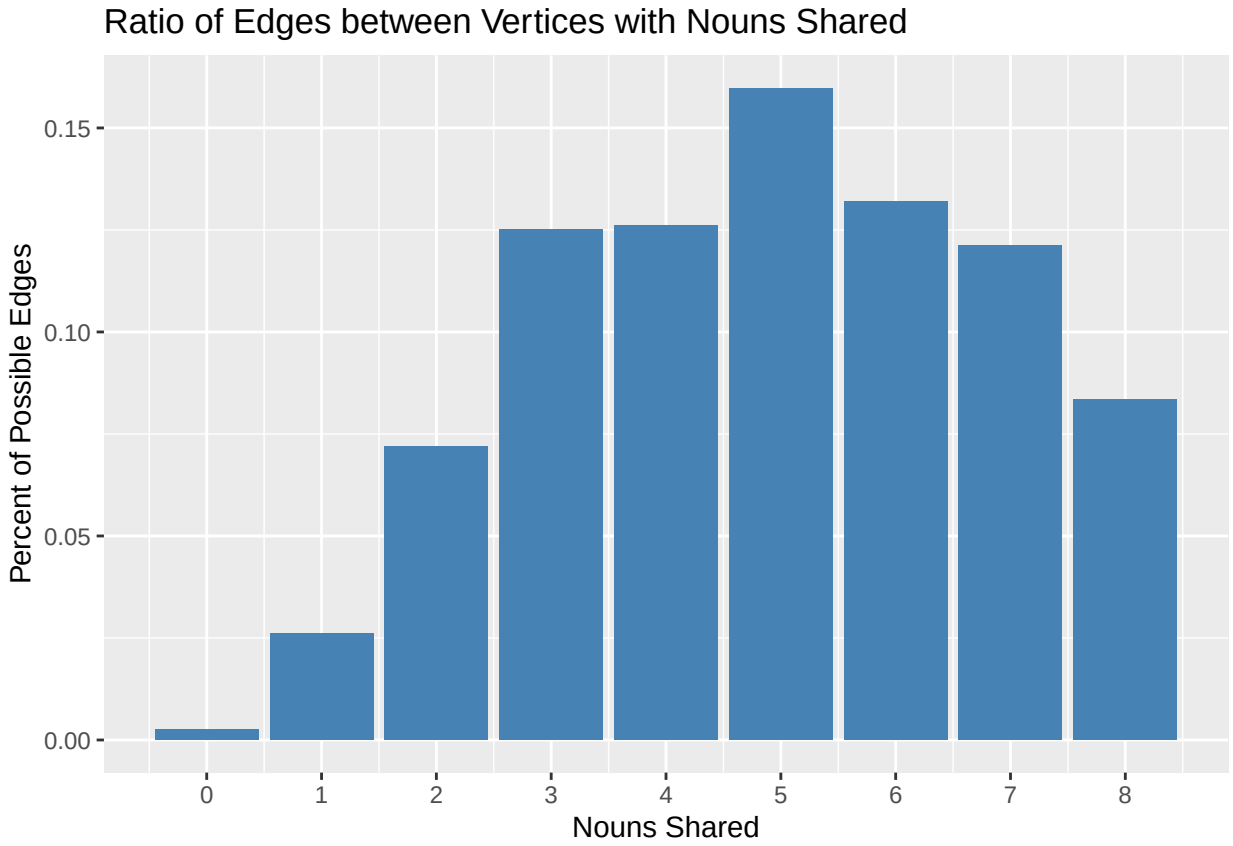
The above table and histogram measures the total number of possible edge combinations between vertices which share a certain amount of nouns, and how many edges actually exist. While at first glance it seems there is a 100% conversion rate for 11 nouns shared to edge possibility, there are only two possible edges, so this is likely insignificant. We can see though that about there is about a 20% edge conversion rate between vertices which share 3, 4 and 5 nouns. From this plot, we can conclude that for this subnetwork, there is about a 20% chance two nodes have an edge if they share between 3 and 5 nouns.

Will: Crocodile

```
## [1] 13530
```

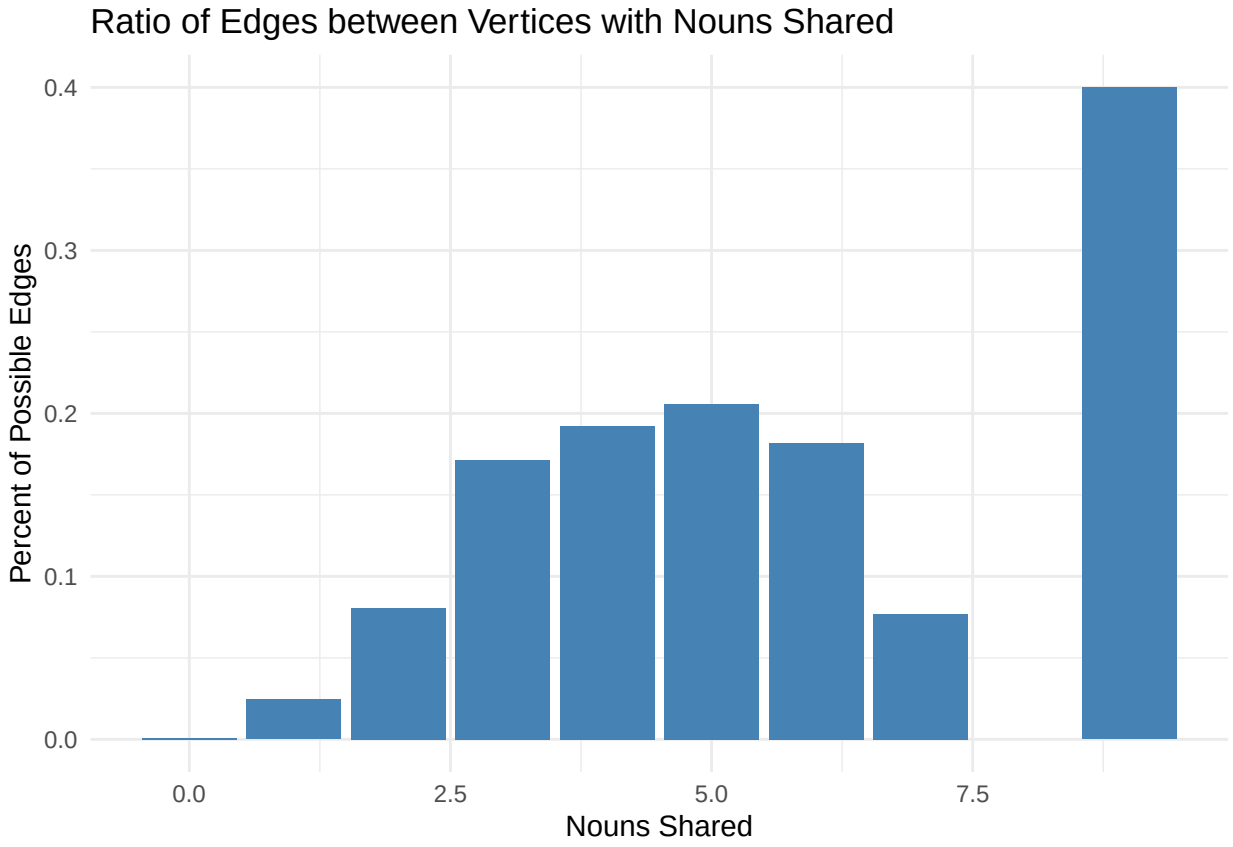
```
##
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   17
## 8325 2409 1347  704  373  194   53   33   12   18   25   19    6    9    1    1
##   19
##    1
```

The table the distribution of the nouns shared by pairs of vertices in the network. We can see that a bulk of the pairs have less than five nouns in common, but there are a significant number of pairs for higher numbers of nouns shared too, going all the way up to 46 (with 41, 44, and 45 missing). Since the network is low density, these percentages will likely be low. But the percentages relative to each other should indicate any relationship in nouns shared.



The barplot shows all of the nonzero percentages for nouns shared (note 9 and 10 do have a percentage of 0 but were left in by the design of our visual). The barplot shows that the highest percentage of a link occurs when two webpages share five nouns with other amounts of nouns in common share similar, but slightly smaller percentages. This, combined with the low density of the network, indicates that sharing nouns with another webpage can boost the chances of there being a connection but that the link is solely based off of this comparison.

Sam: Chameleon




```
##    Amount of Shared Nouns        ratio
## 1                       0 0.0009248555
## 2                       1 0.0244988864
## 3                       2 0.0805604203
## 4                       3 0.1711956522
## 5                       4 0.1921182266
## 6                       5 0.2056074766
## 7                       6 0.1818181818
## 8                       7 0.0769230769
## 10                      9 0.4000000000
```

This graph plots the amount of nouns or keywords shared between two nodes on the x-axis and the percent of possible edges on the y-axis. This bar graph suggests that there is a particularly strong correlation between nodes with two or more shared nouns and the percent of edges between those given nodes. This correlation becomes even stronger as we increase the amount of nouns shared past two. Analysis suggests that there is a correlation between keywords and hyperlinks for Wikipedia articles.

Conclusion

This project gave us insight into how to investigate networks and how to approach interpreting metric statistics. The attributes from our data set allowed us to investigate the relationship between node attributes and how the contribute to the overall structure of the network, as well as contextualize metrics.

From the analysis we conducted, there does not seem to be a correlation between noun count and degree, however there does seem to be a positive correlation between target values and degree, especially within the squirrel network. All three networks also have high variance in the betweenness centrality, meaning

contextually for one to travel from page to page in the network using hyperlinks, there are a few common pages they will likely have to travel thorugh to get anywhere. Additionally, in the crocodile network, there were few edges which crossed community in community detection, meaning the network is highly separated, ie, its difficult to travel across the pages using hyperlinks, as you will likely get caught in a community which doesn't have an obvious pathway out to another community. The opposite was true in the squirrel network. Lastly, for all three animal subnetworks it was the case that 3 to 5 nouns being shared between two vertices meant those two vertices had about a 20% chance of an edge between them. Anything higher or lower than that had a drop off for all three animals. In the context of Wikipedia, this means that if two pages in these animal subnetworks share two nouns, there's a 20 percent chance there is a hyperlink in one of them to another.

Regression and classification models using machine learning could provide a more accurate guess on how many nouns it takes before nodes are connected. Additionally, performing the same analysis on a larger sub-network could reveal more interesting structures given the low density of the crocodile network.

Additionally, for future analysis it would be interesting to see if there is correlation between target values and specific nouns, as well as if there is a correlation between cross community edges and node attributes. While not possible using the current dataset, it would be interesting to actually assign Wikipedia pages and literal words to the nodes and features attributes respectively. It would also be interesting to combine all three datasets into one and analyze it, but this is not possible as nodes and nouns are indexed independently for each dataset. Lastly, a directed graph would have been more interesting to work with, but alas, the contents of the read.me file say otherwise.

References

Reference 1 Dataset: We used the SNAP Wikipedia Article Networks Dataset

Reference 2 Chat GPT helped us visualize the bar graphs.

Reference 3 Our TA Isaiah Katz, PSTAT PhD candidate at University of California Santa Barbara helped us create the adjacency matrices which re-ordered nodes based on degree and fast_greedy community.

Appendices

Lucas's Code

```
library(igraphdata)
library(igraph)
library(Matrix)
library(dplyr)
library(rjson)
library(ggplot2)

## Some setup so plots are bigger with no margins
#default margins
def_marg <- c(5.1, 4.1, 4.1, 2.1)

#no margins
no_marg <- c(0, 0, 0, 0)
```

```
## load the data: Lucas
squirrel_edges_df <- read.csv("./data/squirrel/musae_squirrel_edges.csv")
squirrel_edges_df <- squirrel_edges_df + 1
squirrel_edges_df
```

```r
squirrel_target_df <- read.csv("./data/squirrel/musae_squirrel_target.csv")
squirrel_target_df$id <- squirrel_target_df$id + 1
squirrel_target_df

squirrel_features_list <- fromJSON(file="./data/squirrel/musae_squirrel_features.json")
squirrel_features_list
```

```r
# Making it the edgelist an igraph object and assigning target as a node attribute
squirrel.edgelist <- matrix(unlist(squirrel_edges_df), ncol = 2)
squirrel.igraph <- graph_from_edgelist((squirrel.edgelist))
V(squirrel.igraph)$target <- squirrel_target_df$target
```

```r
# Making a proper dataframe of the json file
features_df <- as.data.frame(matrix(0, ncol = 2, nrow = 5201))
features_df
names_vec <- names(squirrel_features_list)
names_to_num <- as.numeric(names_vec)
features_df$V1 <- names_to_num+1
features_df$V2 <- squirrel_features_list[][]
features_df <- arrange(features_df, V1)
features_df
```

```r
# Adding the nouns list to the corresponding vertices.
V(squirrel.igraph)$nouns <- features_df$V2
```

```r
#Creating a subgraph
set.seed(4704)
squirrel.subnetwork <- induced_subgraph(squirrel.igraph, sample(V(squirrel.igraph), 200))
```

```r
#original graph with issues

plot(squirrel.subnetwork,
     vertex.label.cex=0.3,
     vertex.label.family="Helvetica",
     edge.width=0.5,
     edge.arrow.size=0.1,
     layout=layout.auto,
     vertex.size=11)
```

```r
#creating a second subnetwork with less of the extraneous nodes

squirrel.subnetwork2 <- subgraph(squirrel.subnetwork, degree(squirrel.subnetwork) > 2)
squirrel.subnetwork2 <- as.undirected(squirrel.subnetwork2)
squirrel.subnetwork2 <- simplify(squirrel.subnetwork2)
```

```r
E(squirrel.subnetwork2)$color <- 'gray'
V(squirrel.subnetwork2)$color <- 'gray'

recolor1 <- function(network){
for(i in seq(from = 1, to = length(network), by = 1)){
  if(V(network)[i]$target < 25000){
    V(network)[i]$color <- '#d6e3ff'
  }
  else if(V(network)[i]$target < 50000){
    V(network)[i]$color <- '#adc8ff'
  }
```

```
  else if(V(network)[i]$target < 75000){
    V(network)[i]$color <- '#9bb4e5'
  }
  else if(V(network)[i]$target < 100000){
    V(network)[i]$color <- '#677899'
  }
}
  return(network)
}

squirrel.subnetwork.targetcolor <- recolor1(squirrel.subnetwork2)

plot(squirrel.subnetwork.targetcolor,
     vertex.label.cex=0.3,
     vertex.label.family="Helvetica",
     edge.width=0.5,
     edge.arrow.size=0.1,
     layout=layout_in_circle(squirrel.subnetwork.targetcolor, order = order(V(squirrel.subnetwork.targe
     vertex.size=11)
```

```
E(squirrel.subnetwork2)$color <- 'gray'
V(squirrel.subnetwork2)$color <- 'gray'
V(squirrel.subnetwork2)$colorweight <- 5

recolor2 <- function(network){
for(i in seq(from = 1, to = length(network), by = 1)){
  if(length(V(network)[[i]]$nouns) < 15){
    V(network)[i]$color <- '#d67088'
    V(network)[i]$colorweight <- 1
  }
  else if(length(V(network)[[i]]$nouns) < 25){
    V(network)[i]$color <- '#c94161'
    V(network)[i]$colorweight <- 2
  }
  else if(length(V(network)[[i]]$nouns) < 35){
    V(network)[i]$color <- '#bc123a'
    V(network)[i]$colorweight <- 3
  }
  else if(length(V(network)[[i]]$nouns) < 45){
    V(network)[i]$color <- '#960e2e'
    V(network)[i]$colorweight <- 4
  }
}
  return(network)
}

squirrel.subnetwork.nouncolor <- recolor2(squirrel.subnetwork2)

plot(squirrel.subnetwork.nouncolor,
     vertex.label.cex=0.3,
     vertex.label.family="Helvetica",
     edge.width=0.5,
     edge.arrow.size=0.1,
     layout=layout_in_circle(squirrel.subnetwork.nouncolor, order = order(V(squirrel.subnetwork.nouncol
```

```
     vertex.size=11)
```

```
#Adjacency Matrix Start
squirrel.adjacency1 <- as_adj(squirrel.subnetwork2)
image(Matrix(squirrel.adjacency1))
```

```
#Reordered based on degree
squirrel.subnetwork2.degree <- degree(squirrel.subnetwork2)
squirrel.subnetwork2.sorted <- order(squirrel.subnetwork2.degree, decreasing = TRUE)
squirrel.subnetwork2.adj.matrix <- as.matrix(as_adjacency_matrix(squirrel.subnetwork2))
squirrel.subnetwork2.sorted.adj <- squirrel.subnetwork2.adj.matrix[squirrel.subnetwork2.sorted, squirrel
image(Matrix(squirrel.subnetwork2.sorted.adj))
```

```
#Metrics and Analysis
is_connected(squirrel.subnetwork2)

articulation_points(squirrel.subnetwork2) #nodes which would make the network disconnected

degree(squirrel.subnetwork2, v=V(squirrel.subnetwork2)[58])
degree(squirrel.subnetwork2, v=V(squirrel.subnetwork2)[68])

V(squirrel.subnetwork2)[58]$target
V(squirrel.subnetwork2)[68]$target

V(squirrel.subnetwork2)[58]$nouns
V(squirrel.subnetwork2)[68]$nouns

diameter(squirrel.subnetwork2) #distance between two farthest apart nodes

transitivity(squirrel.subnetwork2) #probability two nodes are connected

mean_distance(squirrel.subnetwork2) #average distance between two nodes

degree(squirrel.subnetwork2, v=V(squirrel.subnetwork2)[36]) #degree of the edge with the most nouns

#betweenness centrality
max(betweenness(squirrel.subnetwork2, weights = NA))
which.max(betweenness(squirrel.subnetwork2, weights = NA))
mean(betweenness(squirrel.subnetwork2, weights = NA))
sqrt(var(betweenness(squirrel.subnetwork2, weights = NA)))

mean(V(squirrel.subnetwork2)$target)
sqrt(var(V(squirrel.subnetwork2)$target))

mean(length(V(squirrel.subnetwork2)$nouns))

betweenness(squirrel.subnetwork2, v=V(squirrel.subnetwork2)[36])

edge_density(squirrel.subnetwork2)

statsname_vector <- c('Articluation Points', 'AP Degree', 'AP Target', 'AP Nouns', 'Diameter', 'Transit

stats_vector <- c('58, 68', '63, 3', '126190, 20636', '20, 46', as.character(diameter(squirrel.subnetwo
```

```r
stats_matrix <- cbind(stats_vector)
rownames(stats_matrix) <- statsname_vector
colnames(stats_matrix) <- 'Metrics'
stats_matrix
```

```r
#Community Detection Algorithms
edge_between_cluster <- cluster_edge_betweenness(squirrel.subnetwork2)
modularity(edge_between_cluster)

fast_greedy_cluster <- cluster_fast_greedy(squirrel.subnetwork2)
modularity(fast_greedy_cluster)
```

```r
# adjacency matrix for fast_greedy
g.community <- cluster_fast_greedy(squirrel.subnetwork2)
g.membership <- membership(g.community)
g.sorted <- order(g.membership)

# adjacency matrix construction
g.adj.matrix <- as.matrix(as_adjacency_matrix(squirrel.subnetwork2))
g.sorted.adj <- g.adj.matrix[g.sorted, g.sorted]
image(Matrix(g.sorted.adj))
```

```r
#Community Detection Graphs
plot(edge_between_cluster, squirrel.subnetwork2, vertex.label.cex=0.0001, vertex.size=7,
     edge.arrow.size=0.1, layout=layout.auto)
igraph_options(auto.print.lines = Inf)
edge_between_cluster

plot(fast_greedy_cluster, squirrel.subnetwork2, vertex.label.cex=0.0001, vertex.size=7,
     edge.arrow.size=0.1, layout=layout.auto)
fast_greedy_cluster
```

```r
#Edges which are cross communities for between_cluster
# Get the community membership for each vertex
membership <- membership(edge_between_cluster)

# Initialize a vector to store the indices of edges connecting different communities
inter_community_edges1 <- c()

# Iterate over all edges
for (edge in 1:ecount(squirrel.subnetwork2)) {
  # Get the vertices of the edge
  vertices <- ends(squirrel.subnetwork2, edge)

  # Check if the vertices belong to different communities
  if (membership[vertices[1]] != membership[vertices[2]]) {
    # If they belong to different communities, add the edge index to the list
    inter_community_edges1 <- c(inter_community_edges1, edge)
  }
}

length(inter_community_edges1)
```

```r
#Edges which are cross communities for fast_greedy
#Edges which are cross communities for between_cluster
```

```r
# Get the community membership for each vertex
membership <- membership(fast_greedy_cluster)

# Initialize a vector to store the indices of edges connecting different communities
inter_community_edges2 <- c()

# Iterate over all edges
for (edge in 1:ecount(squirrel.subnetwork2)) {
  # Get the vertices of the edge
  vertices <- ends(squirrel.subnetwork2, edge)

  # Check if the vertices belong to different communities
  if (membership[vertices[1]] != membership[vertices[2]]) {
    # If they belong to different communities, add the edge index to the list
    inter_community_edges2 <- c(inter_community_edges2, edge)
  }
}

length(inter_community_edges2)
```

```r
squirrel.subnetwork3 <- recolor2(squirrel.subnetwork2)
E(squirrel.subnetwork3)$color <- 'red'


for(i in seq(from=1, to=length(E(squirrel.subnetwork3)), by=1)){
h <- ends(squirrel.subnetwork3, E(squirrel.subnetwork3)[i])[1]
k <- ends(squirrel.subnetwork3, E(squirrel.subnetwork3)[i])[2]
E(squirrel.subnetwork3)[i]$color <-      ifelse(length(intersect(V(squirrel.subnetwork3)[[h]]$nouns,V(s
}

plot(squirrel.subnetwork3,
     vertex.label.cex=0.3,
     vertex.label.family="Helvetica",
     edge.width=0.5,
     edge.arrow.size=0.1,
     layout=layout.sphere,
     vertex.size=11)

noun_link_count <- sum(E(squirrel.subnetwork3)$color == 'green')

noun_fail_count <- sum(E(squirrel.subnetwork3)$color == 'orange')

noun_link_count/length(E(squirrel.subnetwork3))

noun_fail_count/length(E(squirrel.subnetwork3))
```

```r
#Making Noun Analysis Matrix

combine_table <- t(combn(seq(from=1, to=79), 2))

empty_df <- data.frame(combine_table)

intersect_add <- function(df){
  end_list <- vector("list", 3081)
```

```
  end_list <- lapply(end_list, function(x) c(0))
  for(i in seq(from=1, to=3081, by=1)){
    vec <- df[i,]
    l <- vec[[1]]
    m <- vec[[2]]
    intersect_want <- intersect(V(squirrel.subnetwork2)[[l]]$nouns, V(squirrel.subnetwork2)[[m]]$nouns)
    wanted <- length(intersect_want)
    if(wanted > 0){
    end_list[i] <- wanted}
  }
  return(end_list)
}

total_list <- intersect_add(empty_df)

total_vec <- unlist(total_list)

empty_df$intersect_count <- total_vec

check_edge <- function(df){
  end_list <- vector("list", 3081)
  end_list <- lapply(end_list, function(x) c(0))
  for(i in seq(from=1, to=3081, by=1)){
    vec <- df[i,]
    l <- vec[[1]]
    m <- vec[[2]]
    if(are_adjacent(squirrel.subnetwork2, V(squirrel.subnetwork2)[l], V(squirrel.subnetwork2)[m]) == TRU
      end_list[i] = 1
    }
  }
  return(end_list)
}

yes.no_list <- check_edge(empty_df)

yes.no_vec <- unlist(yes.no_list)
empty_df$edge_exist <- yes.no_vec
empty_df

thing <- seq(from=0, to=max(empty_df$intersect_count), by=1)
matrix_thing <- c(thing, 0,0,0,0,0,0,0,0,0,0,0,0,0,0)

percents_matrix <- matrix(matrix_thing, nrow = 14, ncol = 2)

totals_create <- function(df){
  empty_vec <- rep(0, 14)
  for(i in seq(from=0, to=13, by=1)){
    counter = 0
    for(j in seq(from=1, to=length(df$intersect_count))){
      if(df$intersect_count[j]==i){
        counter = counter + 1
      }
    }
```

```
    empty_vec[i+1] = counter
  }
  return(empty_vec)
}

edge_share_totals <- totals_create(empty_df)

percents_matrix[, 2] <- edge_share_totals

edge_create <- function(df){
  empty_vec <- rep(0, 14)
  for(i in seq(from=0, to=13, by=1)){
    counter = 0
    for(j in seq(from=1, to=length(df$intersect_count))){
      if(df$intersect_count[j]==i){
        if(df$edge_exist[j]==1)
          {counter = counter + 1}
      }
    }
    empty_vec[i+1] = counter
  }
  return(empty_vec)
}

edge_truth_total <- edge_create(empty_df)
percents_matrix <- cbind(percents_matrix, edge_truth_total)
ratio <- edge_truth_total/edge_share_totals
percents_matrix <- cbind(percents_matrix, ratio)
percents_matrix[11,4] <- 0
colnames(percents_matrix)[1] <- 'Amount of Shared Nouns'
colnames(percents_matrix)[2] <- 'Vertice Combinations'
colnames(percents_matrix)[3] <- 'Num of Edges'
percents_matrix
```

```
percents_matrix_to_graph <- percents_matrix
df_to_graph <- percents_matrix_to_graph[,c(1,4)] %>% as.data.frame()

ggplot(df_to_graph, aes(x = `Amount of Shared Nouns`, y = `ratio`)) +
  geom_bar(stat = "identity", fill = "#abc0ca") +
  labs(x = "Nouns Shared", y = "Share of Possible Edges") +
  ggtitle("Ratio of Edges between Vertices with Nouns Shared") +
  theme_minimal() +
  scale_x_continuous(n.breaks = 13, limits=c(0,13)) +
  scale_y_continuous(n.breaks=10, limits=c(0,1))
```

Will's Code

```
# read in edge data
crocodile.edge.data <- read.table("./data/crocodile/musae_crocodile_edges.csv", sep = ',',  header = TRU

# the indexing of the nodes starts with 0 so to change to an igraph object we shifted the indexing of t
crocodile.edgelist <- matrix(unlist(crocodile.edge.data), ncol = 2) + 1
```

```r
# converting edge list matrix to igraph object
crocodile.igraph <- graph_from_edgelist((crocodile.edgelist))

# the source of the data says the graph is un-directed
crocodile.igraph <- as.undirected(crocodile.igraph)

# create the adjacency matrix for the network
crocodile.adjacency <- as_adj(crocodile.igraph)

# read in target data
crocodile.target.data <- read.table("./data/crocodile/musae_crocodile_target.csv",
                                    sep = ',',
                                    header = TRUE,
                                    stringsAsFactors = TRUE)

# add one so indices line up with node values
crocodile.target.data$id <- crocodile.target.data$id + 1

# add target values as node attributes
V(crocodile.igraph)$target <- crocodile.target.data$target

# adding name as another node attribute
V(crocodile.igraph)$name <- as.character(crocodile.target.data$id)

# reading in JSON file
crocodile.features <- fromJSON(file = "./data/crocodile/musae_crocodile_features.json")

# converting JSON file to tibble. data frame doesn't work since we're trying to store a list in our sec
crocodile.features.df <- tibble::tibble(name = names(crocodile.features),
                                        value = crocodile.features)

# change name to numeric values for sorting (sorting as characters would cause problems)
crocodile.features.df$name <- as.numeric(crocodile.features.df$name)

# organizing rows based on index
crocodile.features.df <- crocodile.features.df %>% arrange(name)

# add nouns as node attribute
V(crocodile.igraph)$nouns <- crocodile.features.df$value

# Generating the sub-graph
set.seed(17)

sub.network <- induced_subgraph(crocodile.igraph, sample(V(crocodile.igraph), 200))
```

```r
plot(sub.network,
     vertex.color = 'grey',
     vertex.label.family="Helvetica",
     vertex.label.cex = 0.5,
     vertex.label.color = 'black',
     edge.color = "blue")
```

```r
# create subgraph with nodes with degree > 0
temp <- subgraph(sub.network, degree(sub.network) > 0)
```

```
temp <- simplify(temp)

# coloring vertices based of the target data
V(temp)$color <- 'gray'
E(temp)$color <- 'gray'

recolor.using.target <- function(network){
for(i in seq(from = 1, to = length(network), by = 1)){
  if((V(network)[[i]]$target) < 100){
    V(network)[[i]]$color <- '#d67088'
  }
  else if((V(network)[[i]]$target) < 1000){
    V(network)[i]$color <- '#c94161'
  }
  else if((V(network)[[i]]$target) < 10000){
    V(network)[i]$color <- '#bc123a'
  }
  else if((V(network)[[i]]$target) < 100000){
    V(network)[i]$color <- '#960e2e'
  }
  else {
    V(network)[i]$color <- '#890020'
  }
}
  return(network)
}

temp.recolor <- recolor.using.target(temp)

plot(temp.recolor,
     vertex.label.cex=0.3,
     vertex.label.family="Helvetica",
     edge.width=0.5,
     edge.arrow.size=0.1,
     layout=layout_in_circle(temp.recolor, order = order(V(temp.recolor)$target)),
     vertex.size=11)
```

```
# coloring vertices based of the target data
V(temp)$color <- 'gray'
E(temp)$color <- 'gray'
V(temp)$colorweight <- 0

recolor.using.nouns <- function(network){
for(i in seq(from = 1, to = length(network), by = 1)){
  if(length(V(network)[[i]]$nouns) < 15){
    V(network)[[i]]$color <- 'green'
    V(network)[i]$colorweight <- 1
  }
  else if(length(V(network)[[i]]$nouns) < 35){
    V(network)[i]$color <- 'green2'
    V(network)[i]$colorweight <- 2
  }
  else if(length(V(network)[[i]]$nouns) < 55){
    V(network)[i]$color <- 'green3'
```

```
    V(network)[i]$colorweight <- 3
  }
  else if(length(V(network)[[i]]$nouns) < 75){
    V(network)[i]$color <- 'green4'
    V(network)[i]$colorweight <- 4
  }
  else {
    V(network)[i]$color <- 'darkgreen'
    V(network)[i]$colorweight <- 5
  }
}
  return(network)
}

temp.recolor <- recolor.using.nouns(temp)

plot(temp.recolor,
     vertex.label.cex=0.3,
     vertex.label.family="Helvetica",
     edge.width=0.5,
     edge.arrow.size=0.1,
     layout=layout.circle,
     layout=layout_in_circle(temp.recolor, order = order(V(temp.recolor)$colorweight)),
     vertex.size=11)
```

```
temp.recolor <- recolor.using.nouns(temp)
E(temp)$color <- 'red'

for(i in seq(from=1, to=length(E(temp)), by=1)){
h <- ends(temp, E(temp)[i])[1]
k <- ends(temp, E(temp)[i])[2]
E(temp)[i]$color <-      ifelse(length(intersect(V(temp)[[h]]$nouns,V(temp)[[k]]$nouns)) > 2, 'forestgr
}

plot(temp.recolor,
     vertex.label.cex=0.3,
     vertex.label.family="Helvetica",
     edge.width=0.5,
     edge.arrow.size=0.1,
     layout=layout.sphere,
     vertex.size=11)

noun_link_count <- sum(E(temp)$color == 'forestgreen')

noun_fail_count <- sum(E(temp)$color == 'red')

#noun_link_count/length(E(temp))

#noun_fail_count/length(E(temp))
```

```
degree(temp) %>% table()

edge_density(temp) # same as graph.density
```

```r
transitivity(temp)
```

```r
mean_distance(sub.network) # same as average.path.length

diameter(temp)
```

```r
articulation_points(temp) # articulation points (number based off original network)

max(length(V(temp)$nouns))

max(betweenness(temp)) # highest betweenness centrality

sqrt(var(betweenness(temp))) # std dev of betweenness centrality
```

```r
mean(V(temp)$target) # mean target value

sqrt(var(V(temp)$target)) # std dev of target values
```

```r
is_connected(temp)

components(temp)
```

```r
kc <- cluster_edge_betweenness(temp)

length(kc)
sizes(kc)
modularity(kc)

plot(kc, temp,
     vertex.label.cex = 0.5,
     vertex.size = 10)
```

```r
kc <- cluster_fast_greedy(temp)

length(kc)
sizes(kc)
modularity(kc)

plot(kc, temp,
     vertex.label.cex = 0.5,
     vertex.size = 10)
```

```r
communities <- cluster_fast_greedy(temp)

# Get the community membership for each vertex
membership <- membership(communities)

# Initialize a vector to store the indices of edges connecting different communities
inter_community_edges <- c()

# Iterate over all edges
for (edge in 1:ecount(temp)) {
  # Get the vertices of the edge
  vertices <- ends(temp, edge)
```

```
  # Check if the vertices belong to different communities
  if (membership[vertices[1]] != membership[vertices[2]]) {
    # If they belong to different communities, add the edge index to the list
    inter_community_edges <- c(inter_community_edges, edge)
  }
}

print(inter_community_edges)
```

```
# E(temp)[48] : 8115--10438
length(intersect(V(crocodile.igraph)[[8115]]$nouns,
                 V(crocodile.igraph)[[10438]]$nouns))
```

```
temp.adj.matrix <- as.matrix(as_adjacency_matrix(temp))
image(Matrix(temp.adj.matrix))
```

```
# For degree:
temp.degree <- degree(temp)
temp.sorted <- order(temp.degree, decreasing = TRUE)
temp.adj.matrix <- as.matrix(as_adjacency_matrix(temp))
temp.sorted.adj <- temp.adj.matrix[temp.sorted, temp.sorted]
image(Matrix(temp.sorted.adj))
```

```
temp.community <- cluster_fast_greedy(temp)  # whatever community detection algorithm you prefer can be
temp.membership <- membership(temp.community)  # obtain numeric vector consisting of all group membersh
temp.sorted <- order(temp.membership)  # sorts vertices based on group membership

# adjacency matrix construction
temp.adj.matrix <- as.matrix(as_adjacency_matrix(temp))
temp.sorted.adj <- temp.adj.matrix[temp.sorted, temp.sorted]
image(Matrix(temp.sorted.adj))
```

```
# table to represent every vertex combination in the temp network
combo_table <- t(combn(seq(from = 1, to = length(V(temp))), 2))

length(combo_table[,1])

# using dimensions of combo_table to add data
empty_df <- data.frame(combo_table)

# function to count number of shared nouns
intersect_add <- function(df){
  end_list <- vector("list", length(combo_table[,1]))
  end_list <- lapply(end_list, function(x) c(0))
  for(i in seq(from = 1, to = length(combo_table[,1]), by = 1)){
    vec <- df[i,]
    l <- vec[[1]]
    m <- vec[[2]]
    intersect_want <- intersect(V(temp)[[l]]$nouns, V(temp)[[m]]$nouns)
    wanted <- length(intersect_want)
    if(wanted > 0){
    end_list[i] <- wanted}
  }
  return(end_list)
}
```

```
total_list <- intersect_add(empty_df)

total_vec <- unlist(total_list)

empty_df$intersect_count <- total_vec

# function to check whether vertices are connected
check_edge <- function(df){
  end_list <- vector("list", length(combo_table[,1]))
  end_list <- lapply(end_list, function(x) c(0))
  for(i in seq(from = 1, to = length(combo_table[,1]), by = 1)){
    vec <- df[i,]
    l <- vec[[1]]
    m <- vec[[2]]
    if(are_adjacent(temp, V(temp)[l], V(temp)[m]) == TRUE){
      end_list[i] = 1
    }
  }
  return(end_list)
}

yes.no_list <- check_edge(empty_df)

yes.no_vec <- unlist(yes.no_list)

empty_df$edge_exist <- yes.no_vec

table(empty_df$intersect_count) # missing 41, 44, 45
```

```
thing <- seq(from = 0, to = max(empty_df$intersect_count), by = 1)
matrix_thing <- c(thing, rep(0, length(thing)))

percents_matrix <- matrix(matrix_thing, nrow = length(thing), ncol = 2)

totals_create <- function(df){
  empty_vec <- rep(0, length(thing))
  for(i in seq(from = 0, to = length(thing) - 1, by = 1)){
    counter = 0
    for(j in seq(from = 1, to = length(df$intersect_count))){
      if(df$intersect_count[j] == i){
        counter = counter + 1
      }
    }
    empty_vec[i+1] = counter
  }
  return(empty_vec)
}

edge_share_totals <- totals_create(empty_df)

percents_matrix[, 2] <- edge_share_totals

edge_create <- function(df){
```

```
  empty_vec <- rep(0, length(thing))
  for(i in seq(from = 0, to = length(thing) - 1, by = 1)){
    counter = 0
    for(j in seq(from = 1, to = length(df$intersect_count))){
      if(df$intersect_count[j] == i){
        if(df$edge_exist[j] == 1)
          {counter = counter + 1}
      }
    }
    empty_vec[i+1] = counter
  }
  return(empty_vec)
}

edge_truth_total <- edge_create(empty_df)
percents_matrix <- cbind(percents_matrix, edge_truth_total)
ratio <- edge_truth_total/edge_share_totals
percents_matrix <- cbind(percents_matrix, ratio)
colnames(percents_matrix)[1] <- 'Amount of Shared Nouns'
colnames(percents_matrix)[2] <- 'Vertice Combinations'
colnames(percents_matrix)[3] <- 'Num of Edges'

percents_matrix_to_graph <- na.omit(percents_matrix)
df_to_graph <- percents_matrix_to_graph[,c(1,4)] %>% as.data.frame()
df_to_graph <- df_to_graph %>% subset(ratio > 0)
```

```
ggplot(df_to_graph, aes(x = `Amount of Shared Nouns`, y = `ratio`)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  labs(x = "Nouns Shared", y = "Percent of Possible Edges") +
  ggtitle("Ratio of Edges between Vertices with Nouns Shared") +
  scale_x_continuous(n.breaks = 13, limits = c(0,12))
```

Sam's Code

```
library(igraphdata)
library(igraph)
library(Matrix)
library(jsonlite)
library(dplyr)
library(ggplot2)
## Some setup so plots are bigger with no margins
#default margins
def_marg <- c(5.1, 4.1, 4.1, 2.1)

#no margins

no_marg <- c(0, 0, 0, 0)

chameleonEdge.df <- read.csv("./musae_chameleon_edges.csv")
chameleonEdge.df

chamleonFeatureslist <- fromJSON("./musae_chameleon_features.json")
chamleonFeatureslist
```

```
chameleonTarget.df <- read.csv("./musae_chameleon_target.csv")
chameleonTarget.df + 1

vector1 <- chameleonTarget.df$id -1
#CEL <- subset(chameleonEdge.df, select = c(1, 2))
```

```
#create data frame out of JSON file to eventually add to network

features_df <- as.data.frame(matrix(0, ncol = 2, nrow = 2277))

names_vec <- names(chamleonFeatureslist)

names_to_num <- as.numeric(names_vec)

features_df$V1 <- names_to_num + 1

features_df$V2 <- chamleonFeatureslist[][]
features_df <- arrange(features_df, V1)
```

```
#transferring all data frames into the main igraph object
chameleon.edgelist <- matrix(unlist(chameleonEdge.df),ncol=2) + 1

chameleon.igraph <- graph_from_edgelist((chameleon.edgelist))
#transfer Target data frame into main igraph object
V(chameleon.igraph)$target <- chameleonTarget.df$target
V(chameleon.igraph)$nouns <- features_df$V2

str_id <- as.character(vector1-1)
V(chameleon.igraph)$str_id <- str_id
```

```
set.seed(999)

chameleon.edgelist <- matrix(unlist(chameleonEdge.df),ncol=2) + 1
#modifying network to undirected
chameleon.undirected <- as.undirected(chameleon.igraph)

sub.network <- induced_subgraph(chameleon.undirected, sample(V(chameleon.undirected), 200))
plot(sub.network, layout=layout.auto)
#modifying network by removing nodes that only point to themselves. This is a result of selecting nodes
simple.sub <- simplify(sub.network, remove.loops = TRUE, remove.multiple =  TRUE)

#Removing nodes with degree 0
nodeless_network <- subgraph(simple.sub, V(simple.sub)[degree(simple.sub) >= 1])

cham1 <- cluster_fast_greedy(nodeless_network)
plot(cham1, nodeless_network)
```

```
#second community detection
betweenCD <- cluster_edge_betweenness(nodeless_network)

plot(betweenCD, nodeless_network)
```

```
set.seed(999)
```

```
modularity(cham1) #undiret
betweenList <- betweenness(nodeless_network)
sqrt(var(betweenList))
max(betweenList)
#make a plot sorted by target
sum(betweenList)/length(betweenList) #should be betweenness average?
#articulation points + information there degree noun count and target
articulation.points(nodeless_network)
artList <- articulation.points(nodeless_network)
#for(k in seq(from = 1, to = length(artList), by = 1)){
 # degree(V(nodeless_network)[[k]])

#}

targetList <- vector(mode = "numeric", length = length(nodeless_network))
nounCountArtList <- vector(mode = "numeric", length = length(artList))
sum(degree(nodeless_network, V(nodeless_network)[artList]))/length(artList)
for(j in seq(from = 1, to= length(nodeless_network), by  = 1)){
  targetList[j] <- V(nodeless_network)[j]$target
  nounCountArtList[j] <- length(V(nodeless_network)[[j]]$nouns)
}
sum(targetList)/length(targetList)
sqrt(var(targetList))
targetList # make these better
nounCountArtList # make this better too
#V(nodeless_network)[[artList]]$target
#betweenness(artList)
transitivity(chameleon.igraph)
transitivity(nodeless_network)
```

```
v <- 0
for(i in seq(from = 1, to = length(nodeless_network), by = 1)){
  v <- c(length(V(nodeless_network)[[i]]$nouns), v)
  if(length(V(nodeless_network)[[i]]$nouns) == 78){
    index <- i
  }
}
# transitivity to compare
# mean dsitance not needed
#node with highest betweeneness and betweennness average
# standard deviation in relation to modularity
#mean target value and standard deviation
#density of graph vs subnetwork
max(v)
#index
degree(nodeless_network, v = V(nodeless_network)[index]) #node with most nouns
betweenness(nodeless_network, v= V(nodeless_network)[index]) #node with most nouns


#Analysis: Both the network wide metrics like closeness and betweenness and the community wide metrics
# this tells us that most websites have several hyperlinks to similar websites because they are all a s
```

```
set.seed(123)
#adjacency matrix
```

```
chameleon.adjacency <- as_adj(nodeless_network)

head(chameleon.adjacency)

image(Matrix(chameleon.adjacency))
#standard network metrics
#graph.attributes(undChameleon)
#edge.attributes(undChameleon)
#vertex.attributes(undChameleon)
```

```
communities <- cluster_edge_betweenness(nodeless_network)

membership <- membership(communities)

inter_community_edgesList <- c()

for(edge in 1:ecount(nodeless_network)){ #temp here works
  vertices <- ends(nodeless_network, edge)

  if(membership[vertices[1]] != membership[vertices[2]]){
    inter_community_edgesList <- c(inter_community_edgesList, edge)
  }
}
length(inter_community_edgesList)
inter_community_edgesList
```

```
#lucas method between edge cluster
E(nodeless_network)$color <- 'red'
V(nodeless_network)[[2]]$nouns
V(nodeless_network)[[2]]$target
for(i in seq(from=1, to=length(E(nodeless_network)), by=1)){
  h <- ends(nodeless_network, E(nodeless_network)[i])[1]

  k <- ends(nodeless_network, E(nodeless_network)[i])[2]

  E(nodeless_network)[i]$color <- ifelse(length(intersect(V(nodeless_network)[[h]]$nouns,V(nodeless_netw
}
V(nodeless_network)$color <- 'brown'
plot(nodeless_network, vertex.label.cex=0.3, vertex.label.family="Helvetica", edge.width=0.5, edge.arrow

noun_link_count <- sum(E(nodeless_network)$color == 'orange')

noun_fail_count <- sum(E(nodeless_network)$color == 'red')

noun_link_count/length(E(nodeless_network))

noun_fail_count/length(E(nodeless_network))
```

```
E(nodeless_network)$color <- 'gray'
V(nodeless_network)$color <- 'gray'
V(nodeless_network)$colorweight <- 5
recolor2 <- function(network){
  for(i in seq(from = 1, to = length(network), by = 1)){
    if(length(V(network)[[i]]$nouns) < 15){
```

```
      V(network)[i]$color <- '#d67088'
      V(network)[i]$colorweight <- 1 }
    else if(length(V(network)[[i]]$nouns) < 25){
      V(network)[i]$color <- '#c94161'
      V(network)[i]$colorweight <- 2 }
    else if(length(V(network)[[i]]$nouns) < 35){
      V(network)[i]$color <- '#bc123a'
      V(network)[i]$colorweight <- 3 }
    else if(length(V(network)[[i]]$nouns) < 45){
      V(network)[i]$color <- '#960e2e'
      V(network)[i]$colorweight <- 4 } }

  return(network) }
nodeless_network.nouncolor <- recolor2(nodeless_network)

plot(nodeless_network.nouncolor, vertex.label.cex=0.3, vertex.label.family="Helvetica", edge.width=0.5,
```

```
E(nodeless_network)$color <- 'gray'
V(nodeless_network)$color <- '#0b5394'
V(nodeless_network)$colorweight <- 5
recolor3 <- function(network){
  for(i in seq(from = 1, to = length(network), by = 1)){
    if(V(network)[i]$target< 2000){
      V(network)[i]$color <- '#cfe2f3'
      V(network)[i]$colorweight <- 1 }
    else if(V(network)[i]$target < 5000){
      V(network)[i]$color <- '#9fc5e8'
      V(network)[i]$colorweight <- 2 }
    else if(V(network)[i]$target < 10000){
      V(network)[i]$color <- '#6fa8dc'
      V(network)[i]$colorweight <- 3 }
    else if(V(network)[i]$target < 20000){
      V(network)[i]$color <- '#3d85c6'
      V(network)[i]$colorweight <- 4 } }

  return(network) }
nodeless_network.targetcolor <- recolor3(nodeless_network)

plot(nodeless_network.targetcolor, vertex.label.cex=0.3, vertex.label.family="Helvetica", edge.width=0.5
V(nodeless_network)
E(nodeless_network)
```

```
#Isaiah the goat
## igraph object g
set.seed(123)
#adjacency matrix normal

chameleon.adjacency <- as_adj(nodeless_network)

head(chameleon.adjacency)

image(Matrix(chameleon.adjacency))
## adj for Target
nodeless_network.target <-V(nodeless_network)$target
```

```
typeof(V(nodeless_network)$target)
nodeless_network.sorted <- order(nodeless_network.target, decreasing = TRUE)
nodeless_network.adj.matrix <- as.matrix(as_adjacency_matrix(nodeless_network))
nodeless_network.sorted.adj <- nodeless_network.adj.matrix[nodeless_network.sorted, nodeless_network.so
image(Matrix(nodeless_network.sorted.adj))
```

```
nodeless_network.degree <- degree(nodeless_network)
nodeless_network.sorted <- order(nodeless_network.degree, decreasing = TRUE)
nodeless_network.adj.matrix <- as.matrix(as_adjacency_matrix(nodeless_network))
nodeless_network.sorted.adj <- nodeless_network.adj.matrix[nodeless_network.sorted, nodeless_network.so
image(Matrix(nodeless_network.sorted.adj))
```

```
g.community <- cluster_fast_greedy(nodeless_network)  # whatever community detection algorithm you pref
g.membership <- membership(g.community)  # obtain numeric vector consisting of all group memberships (m
g.sorted <- order(g.membership)  # sorts vertices based on group membership

## adjacency matrix construction
g.adj.matrix <- as.matrix(as_adjacency_matrix(nodeless_network))
g.sorted.adj <- g.adj.matrix[g.sorted, g.sorted]
image(Matrix(g.sorted.adj))
```

```
# table to represent every vertex combination in the temp network
combo_table <- t(combn(seq(from = 1, to = length(V(nodeless_network))), 2))
length(combo_table[,1]) # using dimensions of combo_table to add data
empty_df <- data.frame(combo_table) # function to count number of shared nouns
intersect_add <- function(df){
  end_list <- vector("list", length(combo_table[,1]))
  end_list <- lapply(end_list, function(x) c(0))
  for(i in seq(from = 1, to = length(combo_table[,1]), by = 1)){
    vec <- df[i,]
    l <- vec[[1]]
    m <- vec[[2]]
    intersect_want <- intersect(V(nodeless_network)[[l]]$nouns, V(nodeless_network)[[m]]$nouns)
    wanted <- length(intersect_want)
    if(wanted > 0){
      end_list[i] <- wanted
      }
    }
  return(end_list) }
total_list <- intersect_add(empty_df)
total_vec <- unlist(total_list)
empty_df$intersect_count <- total_vec # function to check whether vertices are connected
check_edge <- function(df){ end_list <- vector("list", length(combo_table[,1]))
end_list <- lapply(end_list, function(x) c(0))
for(i in seq(from = 1, to = length(combo_table[,1]), by = 1)){
  vec <- df[i,]
  l <- vec[[1]]
  m <- vec[[2]]
  if(are_adjacent(nodeless_network, V(nodeless_network)[l], V(nodeless_network)[m]) == TRUE){
    end_list[i] = 1
    }
  }
return(end_list) }
yes.no_list <- check_edge(empty_df)
```

```
yes.no_vec <- unlist(yes.no_list)
empty_df$edge_exist <- yes.no_vec
table(empty_df$intersect_count) # missing 41, 44, 45
```

```
thing <- seq(from = 0, to = max(empty_df$intersect_count), by = 1)
matrix_thing <- c(thing, rep(0, length(thing)))
percents_matrix <- matrix(matrix_thing, nrow = length(thing), ncol = 2)
totals_create <- function(df){
  empty_vec <- rep(0, length(thing))
  for(i in seq(from = 0, to = length(thing) - 1, by = 1)){
    counter = 0
    for(j in seq(from = 1, to = length(df$intersect_count))){
      if(df$intersect_count[j] == i){
        counter = counter + 1
        }
      }
    empty_vec[i+1] = counter
    }
  return(empty_vec)
  }
edge_share_totals <- totals_create(empty_df)
percents_matrix[, 2] <- edge_share_totals
```

```
edge_create <- function(df){
  empty_vec <- rep(0, length(thing))
  for(i in seq(from = 0, to = length(thing) - 1, by = 1)){
    counter = 0
    for(j in seq(from = 1, to = length(df$intersect_count))){
      if(df$intersect_count[j] == i){
        if(df$edge_exist[j] == 1) {
          counter = counter + 1
          }
        }
      }
    empty_vec[i+1] = counter
    }
  return(empty_vec)
  }
edge_truth_total <- edge_create(empty_df)
percents_matrix <- cbind(percents_matrix, edge_truth_total)
ratio <- edge_truth_total/edge_share_totals
percents_matrix <- cbind(percents_matrix, ratio)
colnames(percents_matrix)[1] <- 'Amount of Shared Nouns'
colnames(percents_matrix)[2] <- 'Vertice Combinations'
colnames(percents_matrix)[3] <- 'Num of Edges'
```

```
percents_matrix_to_graph <- na.omit(percents_matrix)
df_to_graph <- percents_matrix_to_graph[,c(1,4)] %>% as.data.frame()
df_to_graph <- df_to_graph %>% subset(ratio > 0)
ggplot(df_to_graph, aes(x = `Amount of Shared Nouns`, y = `ratio`)) + geom_bar(stat = "identity", fill =
```

```
percents_matrix_to_graph <- na.omit(percents_matrix)
df_to_graph <- percents_matrix_to_graph[,c(1,4)] %>% as.data.frame()
df_to_graph <- df_to_graph %>% subset(ratio > 0)
```

```
ggplot(df_to_graph, aes(x = `Amount of Shared Nouns`, y = `ratio`)) + geom_bar(stat = "identity", fill =
  df_to_graph
```