University of Victoria

CSC - 501

REPORT (ASSIGNMENT – 3)

GRAPH THEORY

SUBMITTED BY - GROUP B

[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]
[redacted]	[redacted]

Submitted To - Prof. Sean Chester

schester@uvic.ca

SECTION - 1

DATA MODELLING

We are given 2 TSV files each containing the links from source sub-reddit to target sub-reddit. Both the files provide hyperlinks mentioned in body and title of a subreddit.

The properties of the files are given below:

	soc-redditHyperlinks-title.tsv	soc-redditHyperlinks-body
Number of nodes/vertices	54,075	35,776
Number of Edges(unique)	234,792	137,821
Number of relationships	571,928	286,561

Taking in consideration the data model for our given dataset we end up choosing the following four physical representations from the graph theory and concepts which we found are the best fit.

1. **Edge List** - Contains list of all the edges (**unique**) from source to target subreddit as (SOURCE_SUBREDDIT, TARGET_SUBREDDIT) tuple.

2. **Adjacency Matrix** - It is a square matrix with size of **|Vertices x Vertices|** and contains either 0's and 1's where 1 represents adjacent node from a node. Therefore, the adjacency matrix for the file is a **sparse matrix** for the given dataset and of 54075 rows and 54075 columns.

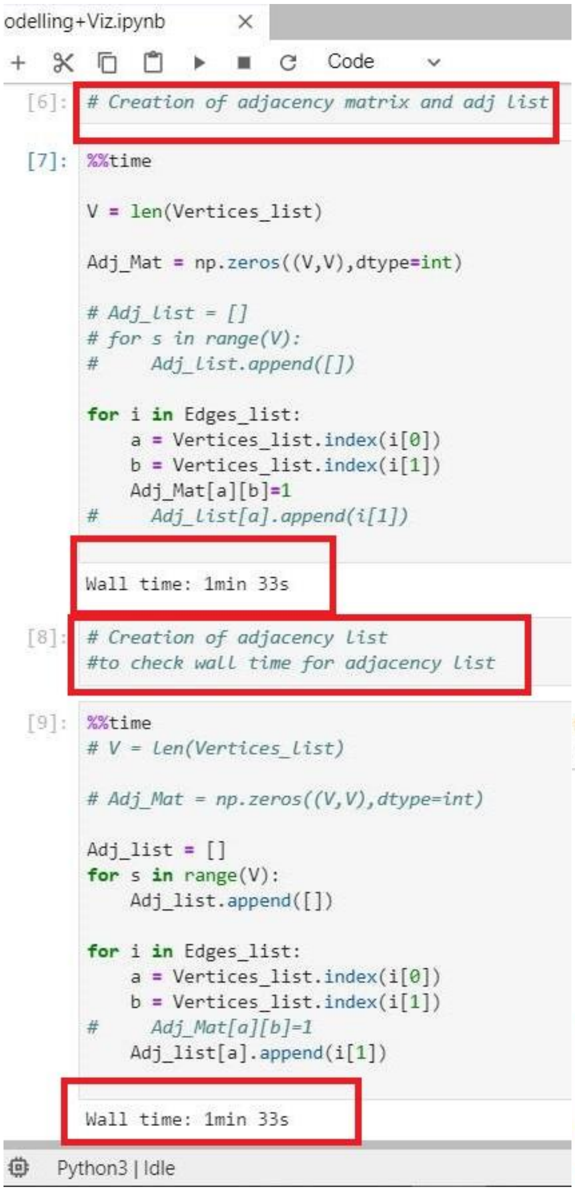
3. **Adjacency List** – It is a list of lists and the length of the adjacency list is same as the count of Vertices, but length of each of the lists within adjacency list depends upon number of adjacent nodes a node has.

4. **Triple Store** – It is property centric and for the given dataset, we took **source node** as **subject**, **post_id** as **predicate** and **target node** as **object**. The relationships are created in **neo4j** by using **CQL** (Cypher Query Language) which took less than 10 seconds for 571928 relationships (soc-redditHyperlinks-title.tsv).

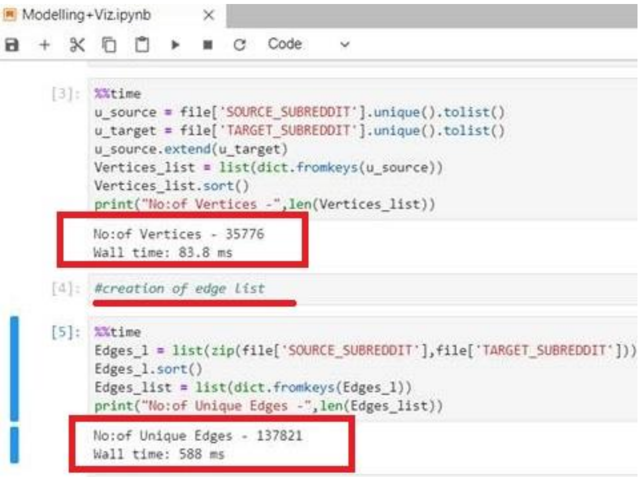
The below are the wall times for the creation of the above data model representation

	Edge-List	Adjacency Matrix	Adjacency List	Triple Store
soc-redditHyperlinks-title.tsv	939ms	7min 16s	7min 56s	9.6s
soc-redditHyperlinks-body	588ms	1min 33s	1min 33s	5.4s

(Screenshots of the above mentioned wall times are provided in the following pages)



```
[6]: # Creation of adjacency matrix and adj list

[7]: %%time

     V = len(Vertices_list)

     Adj_Mat = np.zeros((V,V),dtype=int)

     # Adj_list = []
     # for s in range(V):
     #     Adj_list.append([])

     for i in Edges_list:
         a = Vertices_list.index(i[0])
         b = Vertices_list.index(i[1])
         Adj_Mat[a][b]=1
     #     Adj_list[a].append(i[1])

     Wall time: 1min 33s

[8]: # Creation of adjacency list
     #to check wall time for adjacency list

[9]: %%time
     # V = len(Vertices_list)

     # Adj_Mat = np.zeros((V,V),dtype=int)

     Adj_list = []
     for s in range(V):
         Adj_list.append([])

     for i in Edges_list:
         a = Vertices_list.index(i[0])
         b = Vertices_list.index(i[1])
     #     Adj_Mat[a][b]=1
         Adj_list[a].append(i[1])

     Wall time: 1min 33s
```



```
[3]: %%time
     u_source = file['SOURCE_SUBREDDIT'].unique().tolist()
     u_target = file['TARGET_SUBREDDIT'].unique().tolist()
     u_source.extend(u_target)
     Vertices_list = list(dict.fromkeys(u_source))
     Vertices_list.sort()
     print("No:of Vertices -",len(Vertices_list))

     No:of Vertices - 35776
     Wall time: 83.8 ms

[4]: #creation of edge list

[5]: %%time
     Edges_l = list(zip(file['SOURCE_SUBREDDIT'],file['TARGET_SUBREDDIT']))
     Edges_l.sort()
     Edges_list = list(dict.fromkeys(Edges_l))
     print("No:of Unique Edges -",len(Edges_list))

     No:of Unique Edges - 137821
     Wall time: 588 ms
```

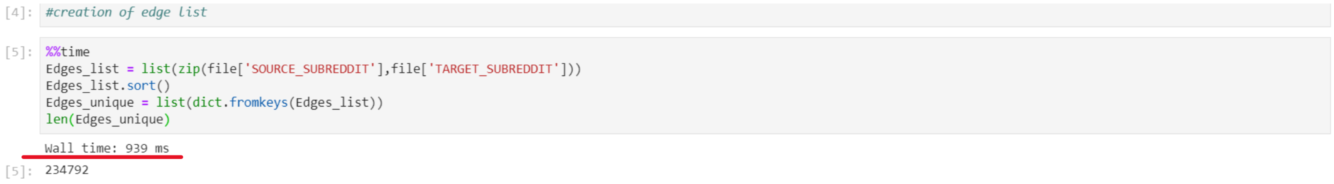
Scenarios to choose respective data models:

- We use an adjacency list if the graph is *sparse* and a adjacency matrix if the graph is *dense*
- Triple store is used when edges are to be retrieved based in relationships

Creation of the above representations:

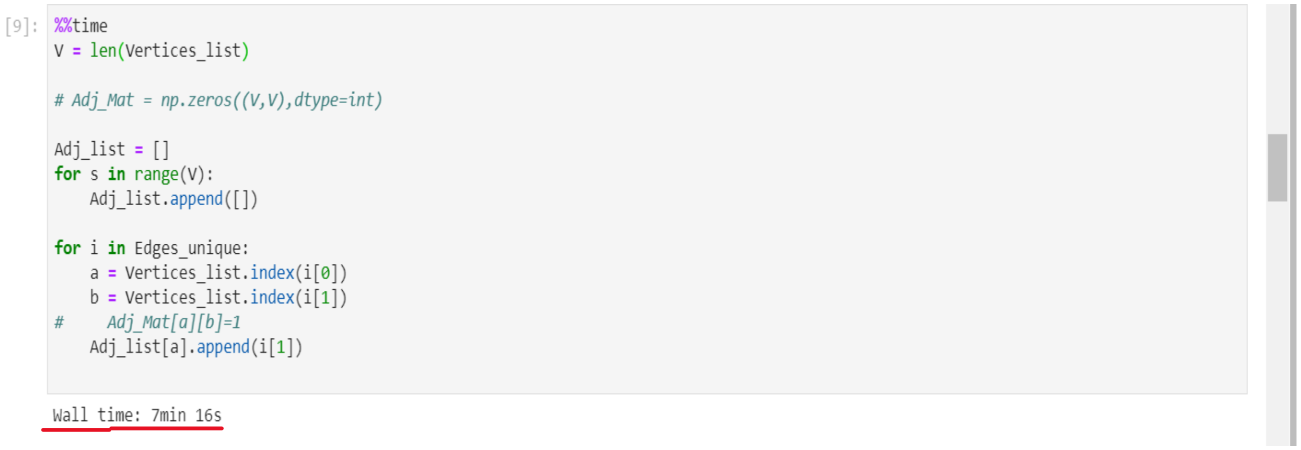
(for file: soc-redditHyperlinks-title.tsv)

Creation of Edge List:



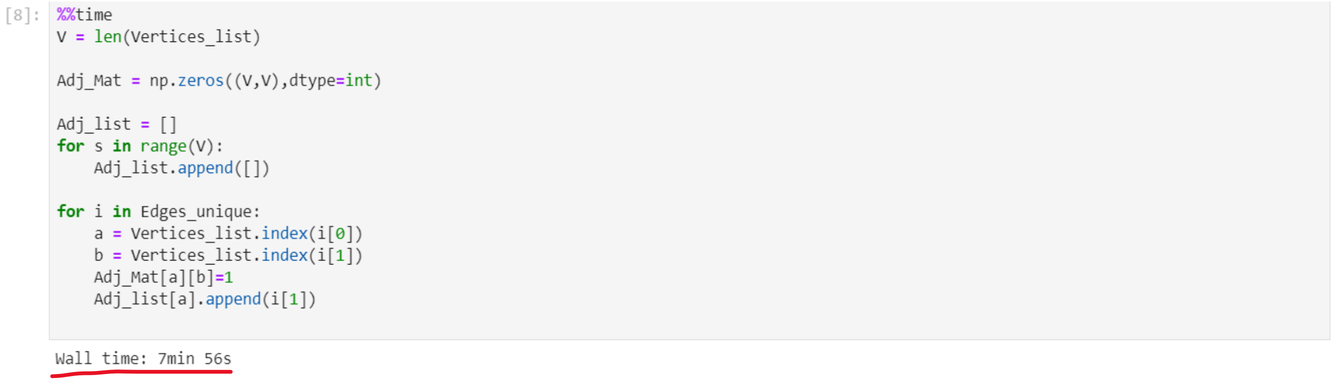
```
[4]: #creation of edge list

[5]: %%time
     Edges_list = list(zip(file['SOURCE_SUBREDDIT'],file['TARGET_SUBREDDIT']))
     Edges_list.sort()
     Edges_unique = list(dict.fromkeys(Edges_list))
     len(Edges_unique)

     Wall time: 939 ms
[5]: 234792
```

Creation of Adjacency Matrix:



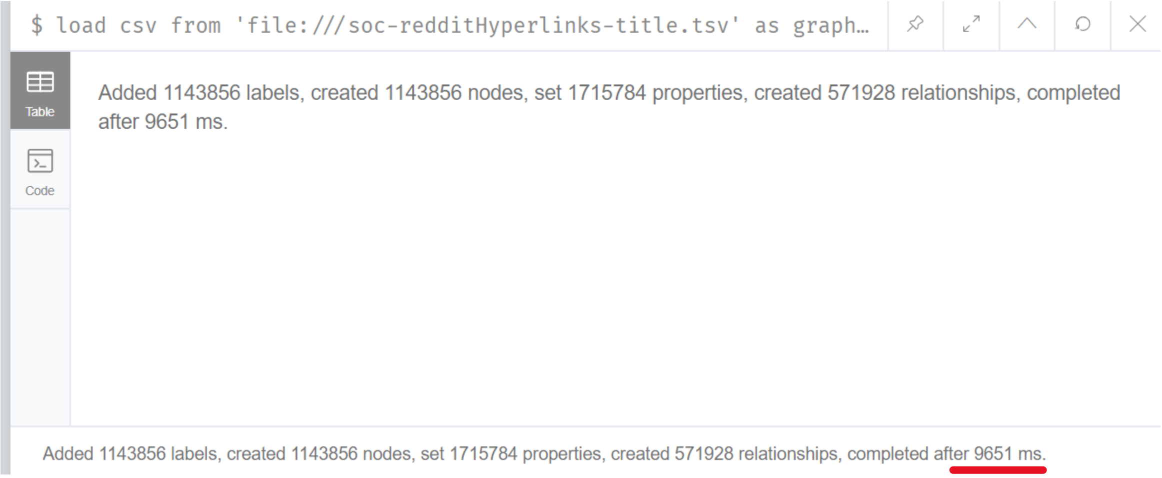
```
[9]: %%time
     V = len(Vertices_list)

     # Adj_Mat = np.zeros((V,V),dtype=int)

     Adj_list = []
     for s in range(V):
         Adj_list.append([])

     for i in Edges_unique:
         a = Vertices_list.index(i[0])
         b = Vertices_list.index(i[1])
     #     Adj_Mat[a][b]=1
         Adj_list[a].append(i[1])

     Wall time: 7min 16s
```

Creation of Adjacency List:



```
[8]: %%time
     V = len(Vertices_list)

     Adj_Mat = np.zeros((V,V),dtype=int)

     Adj_list = []
     for s in range(V):
         Adj_list.append([])

     for i in Edges_unique:
         a = Vertices_list.index(i[0])
         b = Vertices_list.index(i[1])
         Adj_Mat[a][b]=1
         Adj_list[a].append(i[1])

     Wall time: 7min 56s
```

Creation of Triple- store:



```
$ load csv from 'file:///soc-redditHyperlinks-title.tsv' as graph…
```

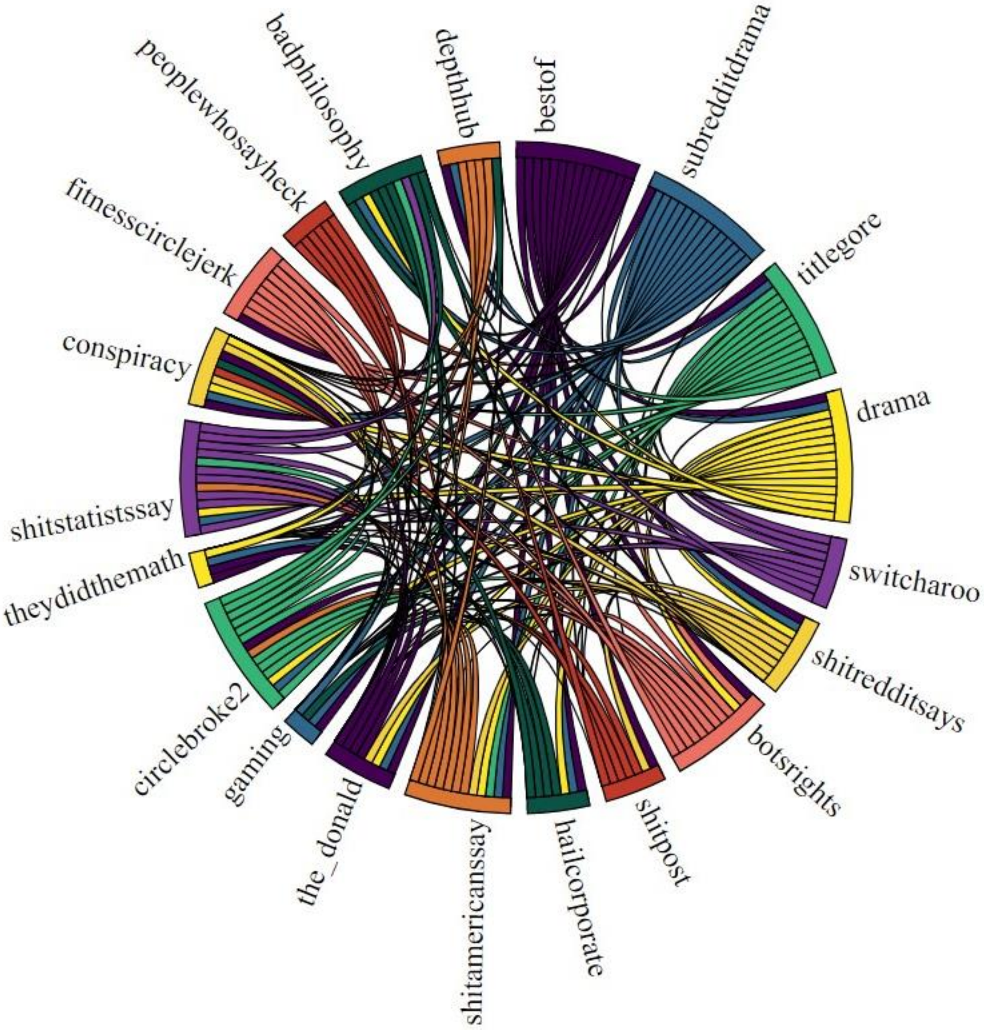
Added 1143856 labels, created 1143856 nodes, set 1715784 properties, created 571928 relationships, completed after 9651 ms.

Added 1143856 labels, created 1143856 nodes, set 1715784 properties, created 571928 relationships, completed after 9651 ms.

Insight using Chord Diagram:

To show how popular subreddits are connected relate to each other, we needed a chord diagram for the insight. The chord diagram is not possible by using d3.js with python and plotting chord diagram in plot.ly was also a complex task. So instead of using these two, we used js and html to plot chord diagram with the data we fetched from our algorithm in python using adjacency list.

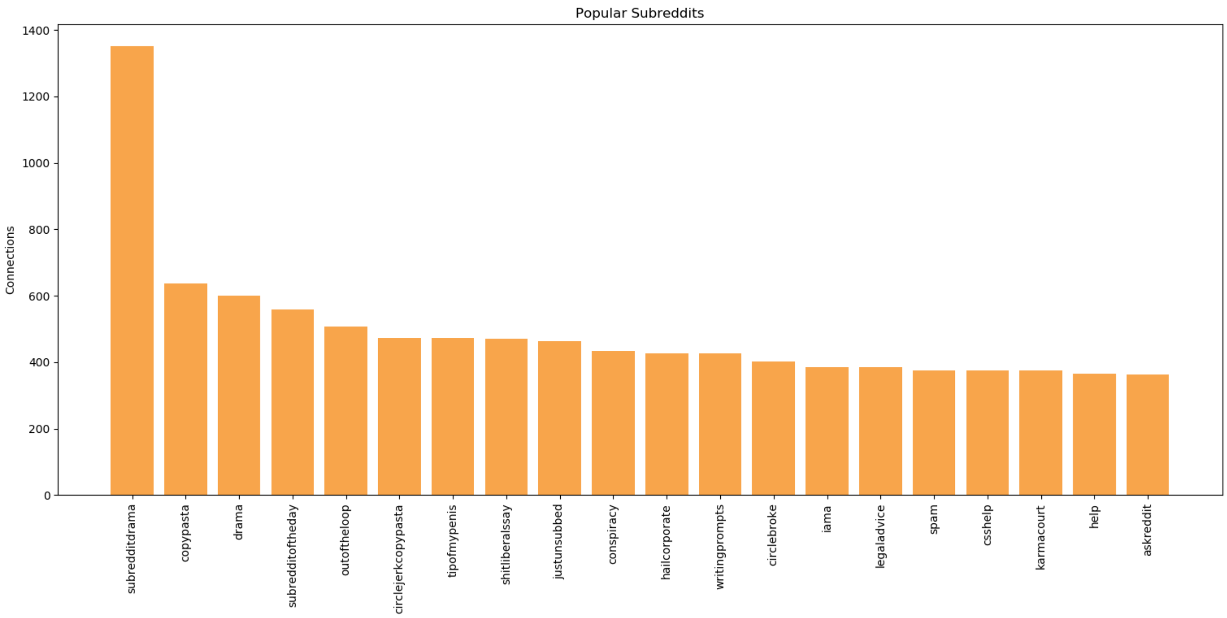
The chord diagram gives the connectivity of top 20 popular nodes among each other (here popularity of a node is taken by considering the nodes with the greatest number of outward edges). The adjacency matrix for the nodes is provided as input for the chord diagram which is implemented by javascript and html.



SECTION - 2

Visualization

INSIGHT #1 : Top 20 popular subreddits and relationship among themselves

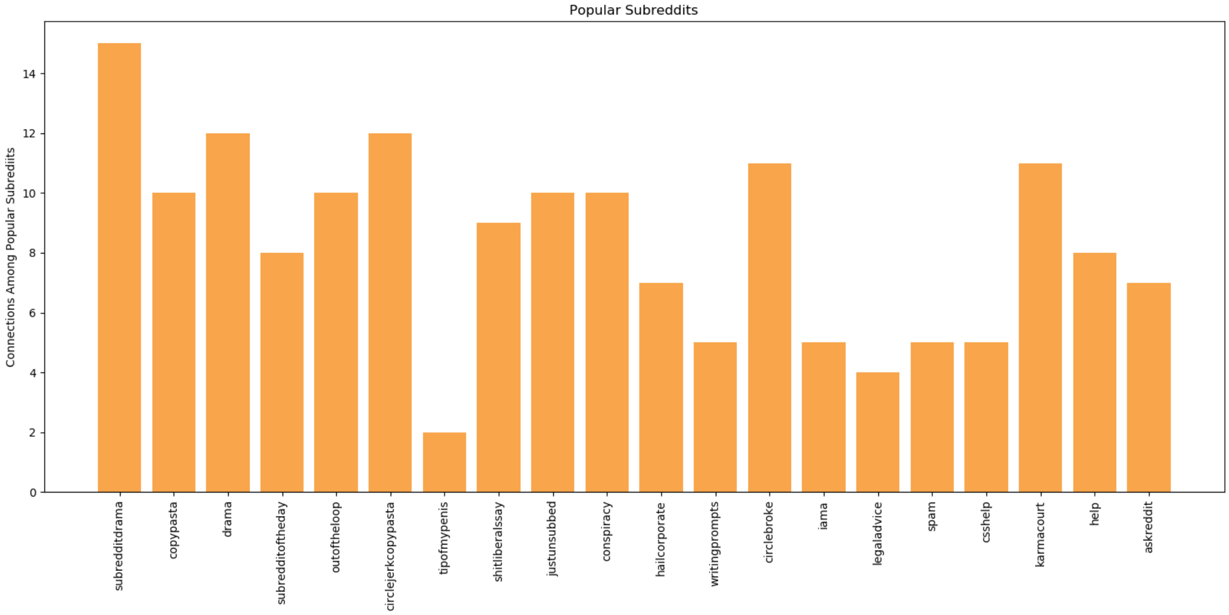


(Here the popularity of a node is taken by considering the node by greatest number of outward edges)

To show how popular subreddits are connected to each other, we needed a chord diagram for the insight. The chord diagram is not possible by using d3.js with python and plotting chord diagram in plot.ly was also a complex task. So instead of using these two, we used js and html to plot chord diagram with the data we fetched from our algorithm in python using adjacency list.

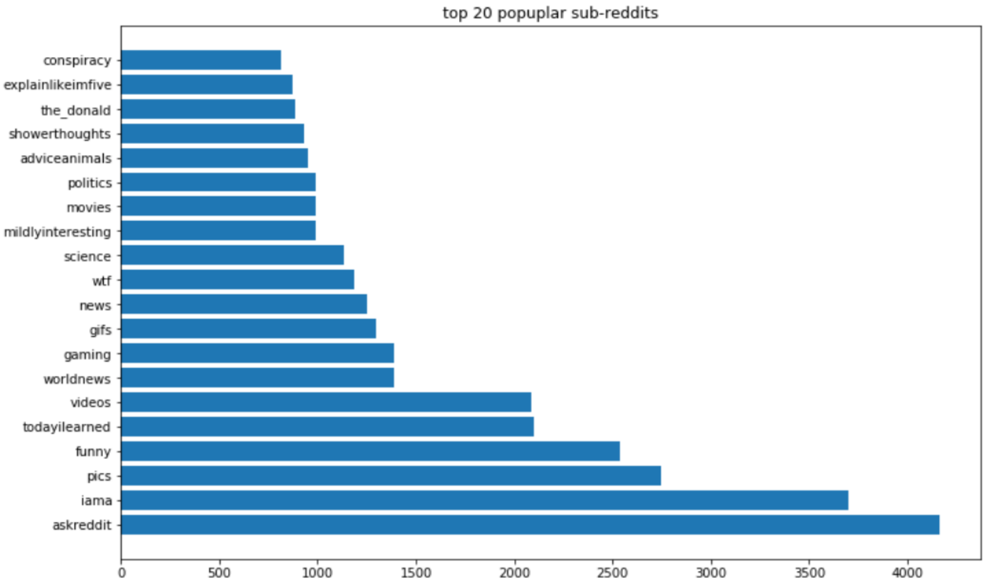
INSIGHT #2 : Subreddits that have the greatest number of responses(node with greatest number of outward edges)–

From the visualization we can derive that **subredditdrama** has exceptionally the maximum number of responses, rest popular subreddit have nearly similar pattern in the number of responses from the other subreddits



INSIGHT #3 : Subreddits that answer the most of the other subreddit (node with greatest number of inward edges)

```
[33]: Text(0.5, 1.0, 'top 20 popuplar sub-reddits')
```



From the visualization we can derive that **askreddit** could be the most directed subreddit for most of the queries, followed closely by **iama**.

INSIGHT #4 : Variation of Subreddit in terms of title and body

4.1 Variation of subreddits in title and body annually.

From the following visualization of number of subreddits in the body and title part, we analyse that both have similar pattern of increasing over the year from 2013 to 2016. For 2017(data is incomplete)

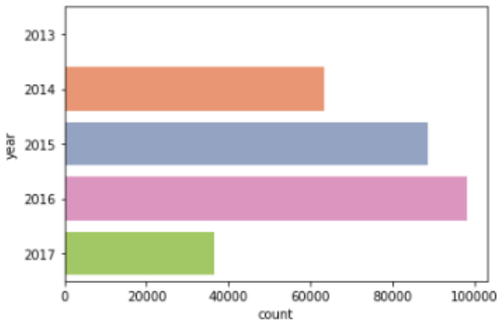


Fig – Subreddit in Body annually

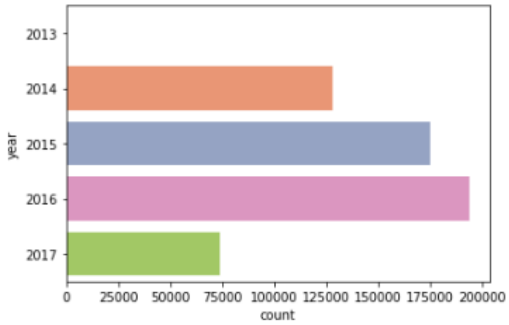
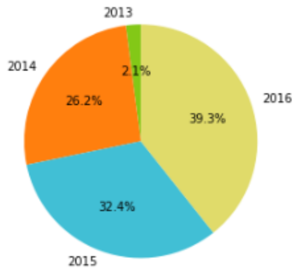
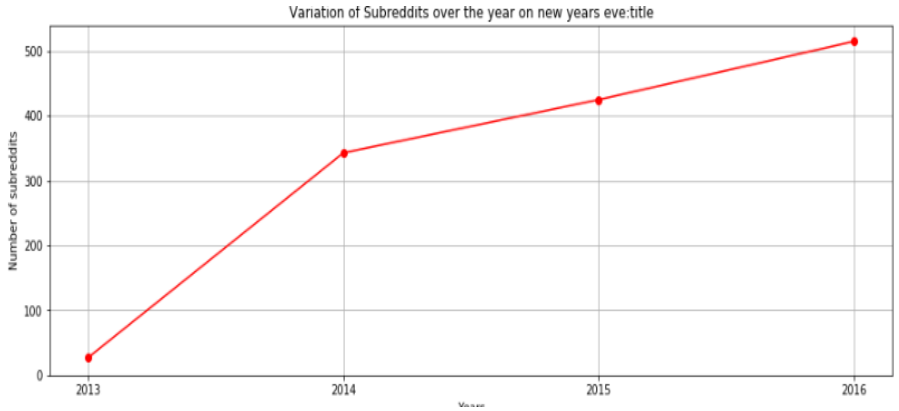


Fig – Subreddit in title annually

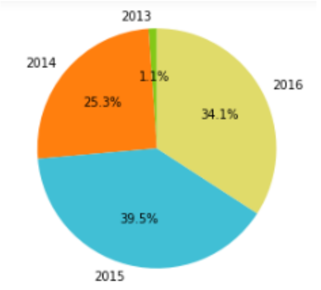
4.2 Variation of subreddits on New Year's Eve (31 Dec) annually.

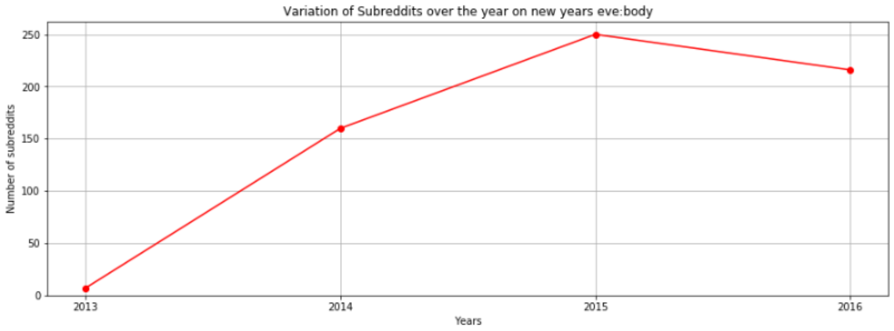
From the visualization we can derive that the number of subreddits increased almost exponentially over the year 2013 to 2016 on the New Year's Eve for titles which were extremely low on the new year eve 2013, eventually marking the highest in 2016 on the same day.





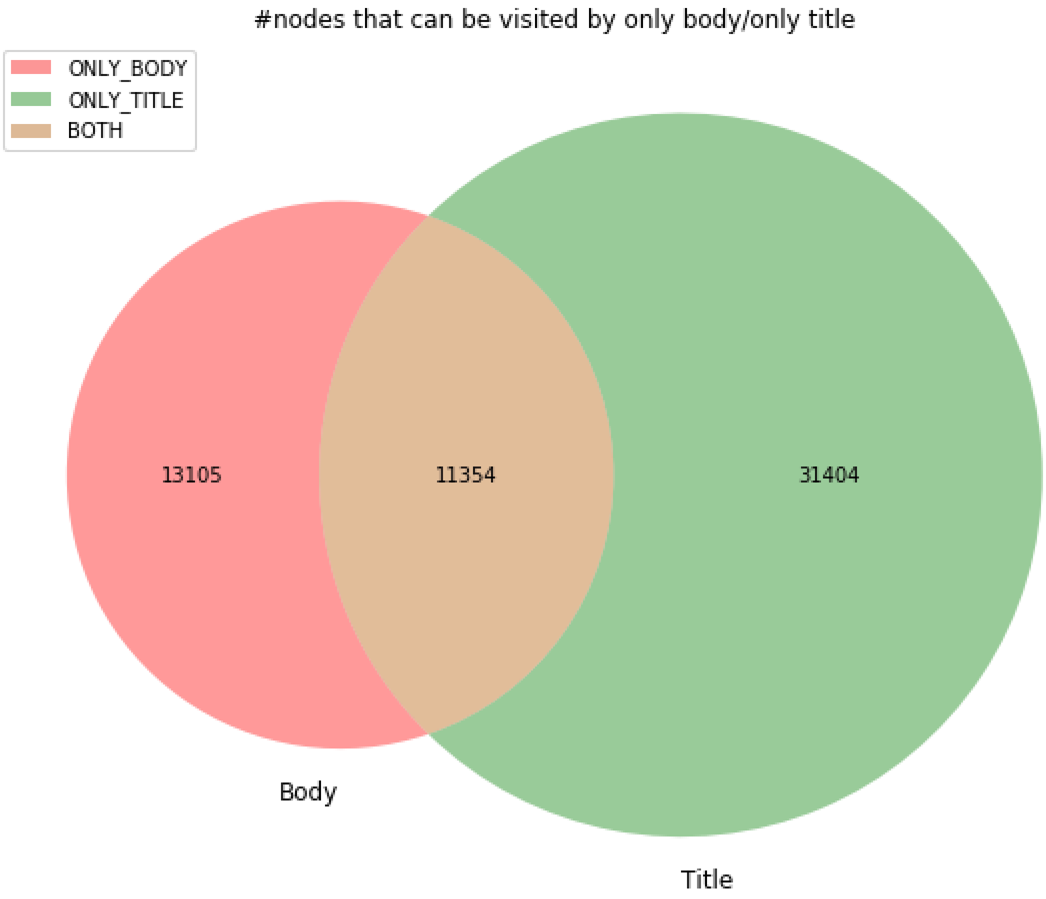
Taking in consideration the body part, the number of subreddits have exponentially increased until 2015 on new years eve, but there is a slight drop in the year 2016 new years eve in no of subreddits for the body.





4.3 Relationship between subreddits in terms of tittle and body

From the visualization we can derive that there are 13105 subreddit that can be only reached through body and 31404 subreddits that can only be reached through titles.

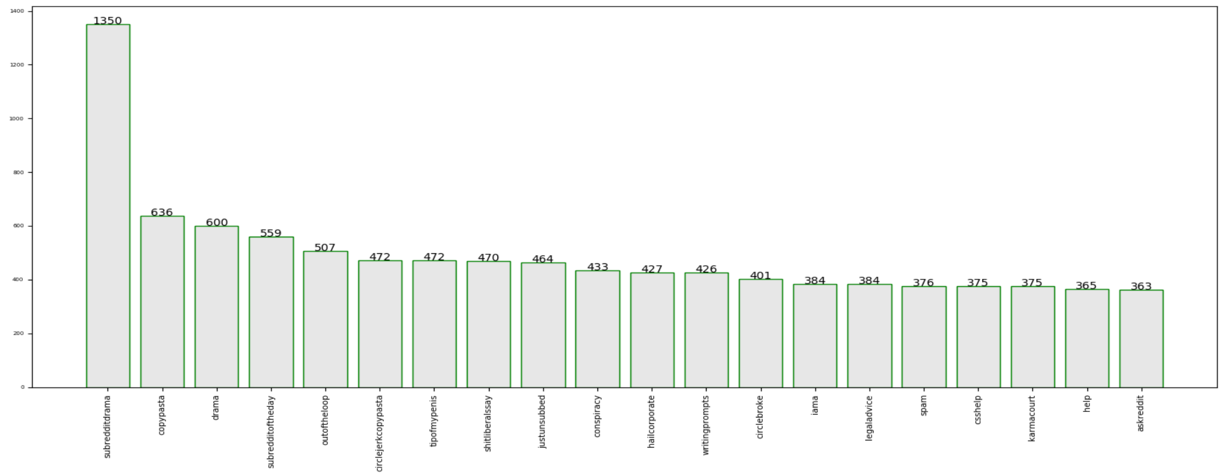


INSIGHT #5 : <u>Top 20 Influential Subreddits</u>

This insight is based on **degree distribution**. We used adjacency list to find the degree of each node(subreddit). The reason behind choosing degree distribution for this insight is because degree distribution is widely used to analyse the most influential nodes in a network. The nodes that have the most connections in a network are the nodes that are influential and have more access to information than the nodes that have few connections. Degree is also a measure of the node's vulnerability of catching a virus or other detrimental information from the network.

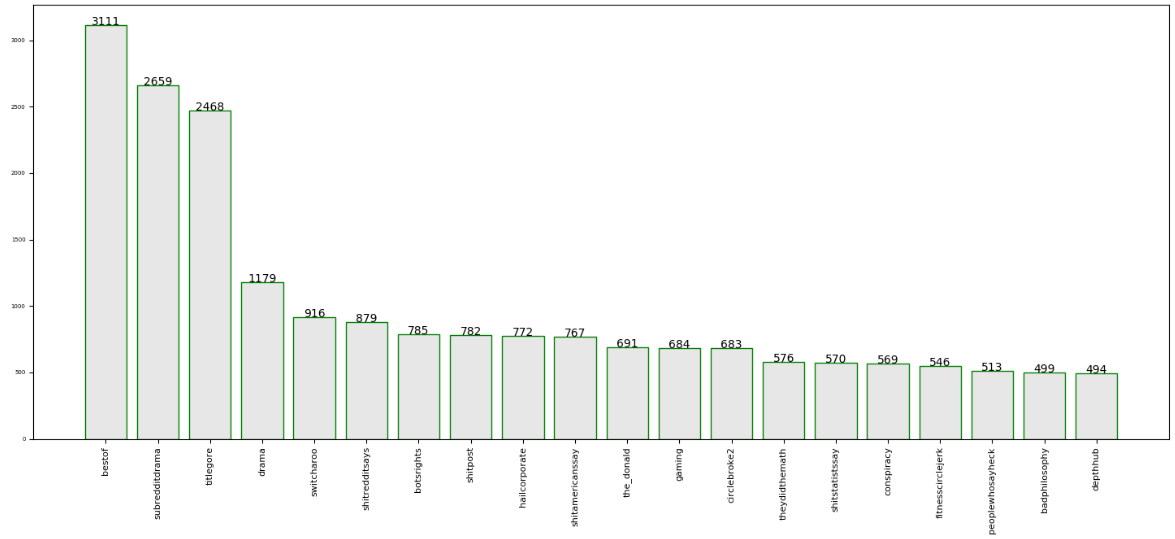
soc-redditHyperlinks-body

From the visualization we can derive that **subredditdrama** is the most influential subreddit influencing 1350 other subreddits



soc-redditHyperlinks-title

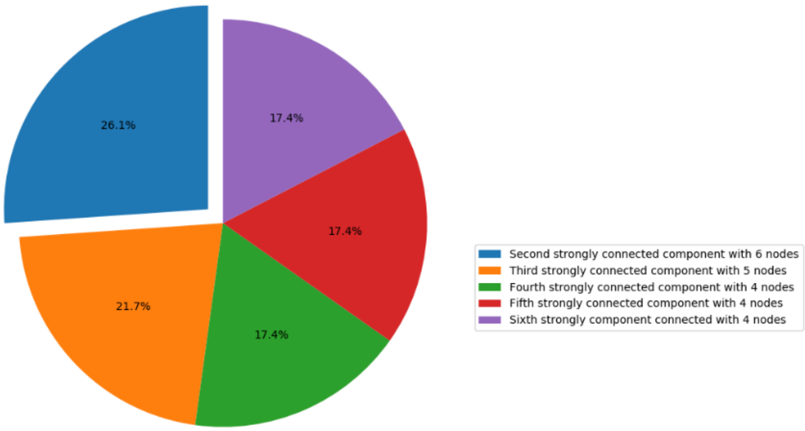
From the visualization we can derive that for title **bestof** is the most influential subreddit influencing 3111 other subreddits closely followed by **subredditdrama** with 2659 influential subreddits(higher than highest in the body),

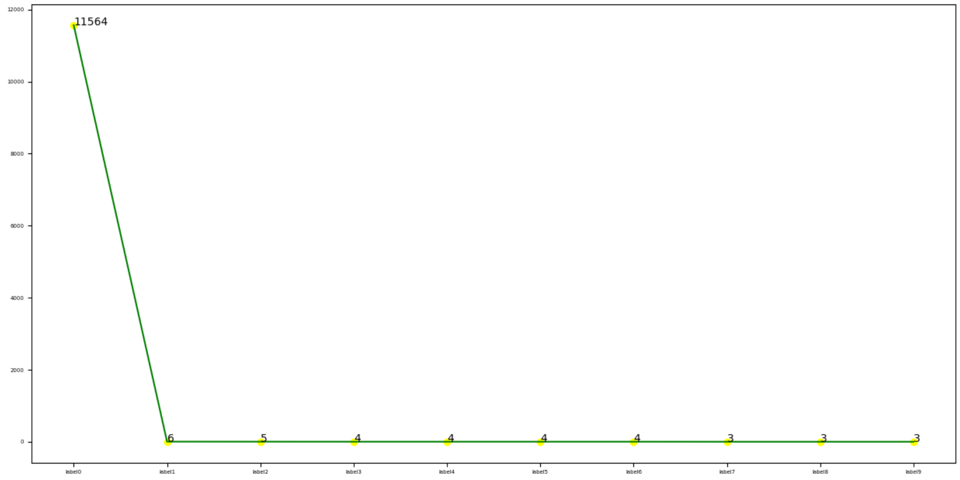


INSIGHT #6: Strongly connected components among the Subreddits

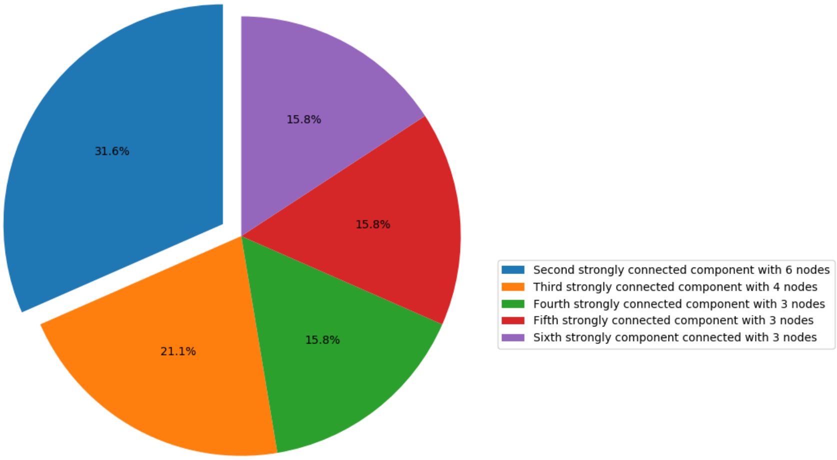
In this insight we are trying to analyse the network with respect to the strongly connected components among the directed Subreddits network. We encountered a giant component (comprising of 11564 subreddits) along with some other smaller components. In a strongly connected component, each vertex can reach every other vertex along a directed path. So, we have a giant component which means each subreddit is connected to every other subreddit in that component. The other strongly connected components are very small compared to the giant component, with the second strongly connected component consisting of only 6 subreddits. Each strongly connected component is completely isolated from the other strongly connected component

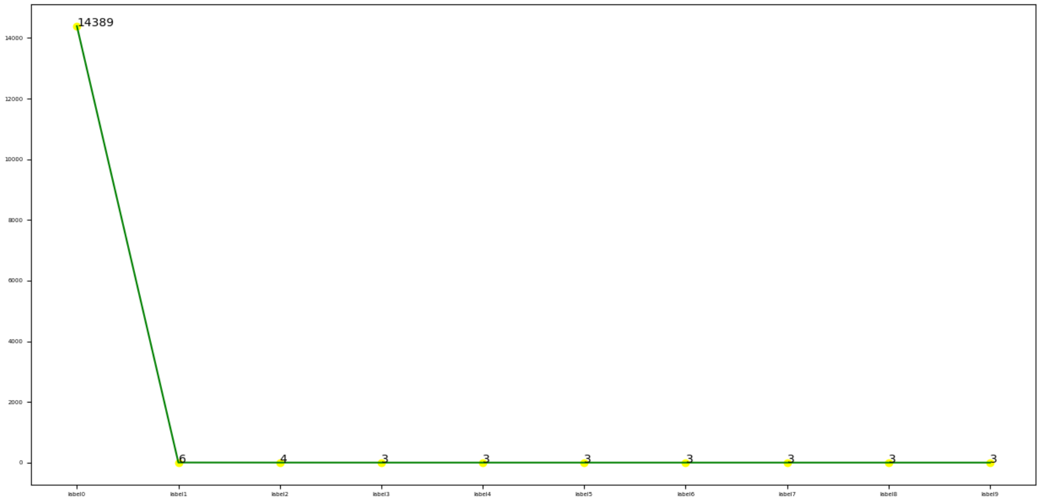
soc-redditHyperlinks-title.tsv





soc-redditHyperlinks-title.tsv

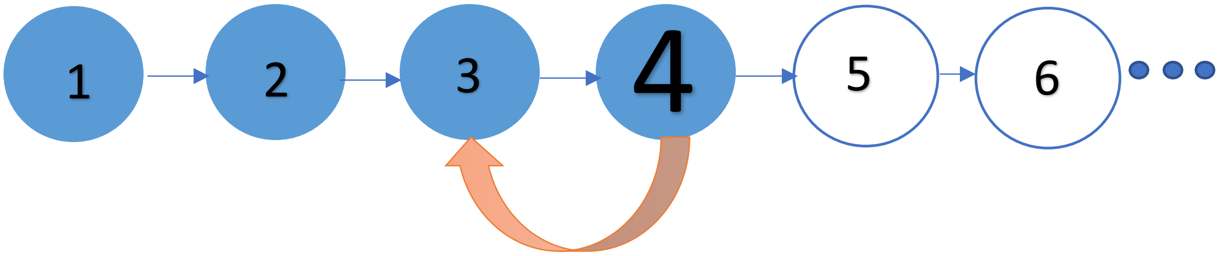

31.6%
15.8%
15.8%
21.1%
15.8%
Second strongly connected component with 6 nodes
Third strongly connected component with 4 nodes
Fourth strongly connected component with 3 nodes
Fifth strongly connected component with 3 nodes
Sixth strongly component connected with 3 nodes

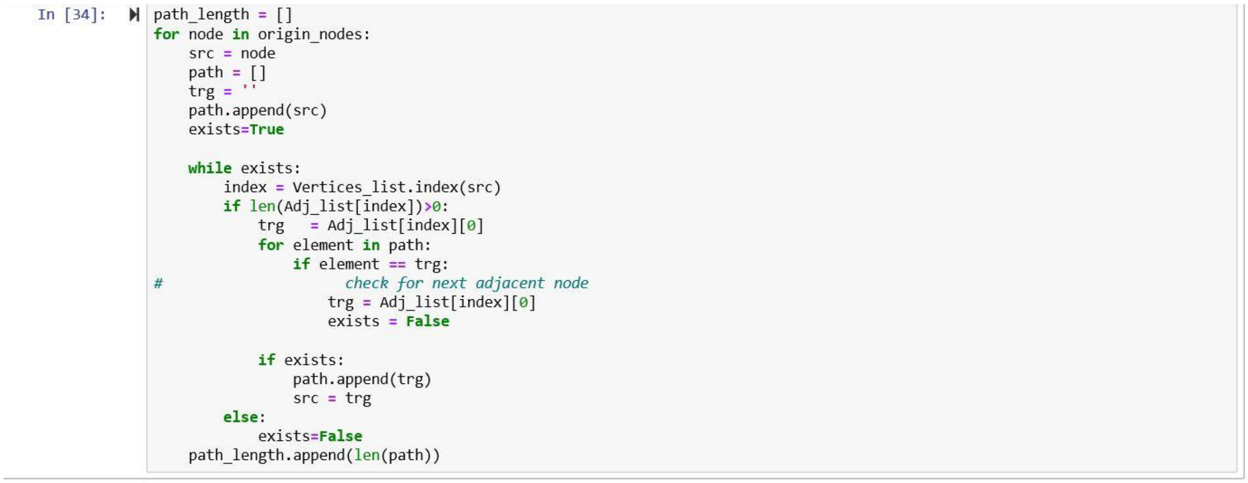

14389
6
4
3
3
3
3
3
3
3
14000
12000
10000
8000
6000
4000
2000
0
label0
label1
label2
label3
label4
label5
label6
label7
label8
label9

INSIGHT #7 : Average path length required to reach a cycle(when source and destination are not same)

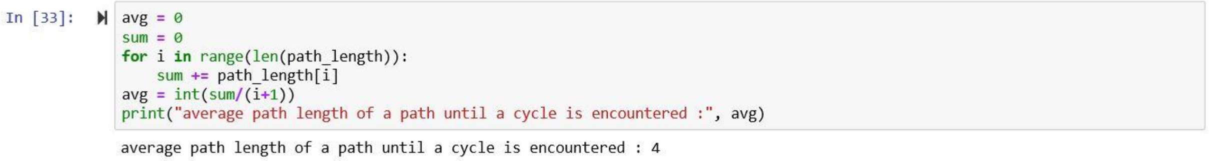
We analysed the number of nodes in a path before forming the first cycle in its path. Calculating the AVERAGE path length of the such paths comes out to be 4 , from which we can derive that from the origin node in a path , it takes 4 nodes to form a cycle.

Therefore, in the diagram below we are considering nodes until ‘4’, after which it forms a cycle.





```
In [34]:  path_length = []
          for node in origin_nodes:
              src = node
              path = []
              trg = ''
              path.append(src)
              exists=True

              while exists:
                  index = Vertices_list.index(src)
                  if len(Adj_list[index])>0:
                      trg    = Adj_list[index][0]
                      for element in path:
                          if element == trg:
          #                     check for next adjacent node
                              trg = Adj_list[index][0]
                              exists = False

                      if exists:
                          path.append(trg)
                          src = trg
                  else:
                      exists=False
              path_length.append(len(path))
```



```
In [33]:  avg = 0
          sum = 0
          for i in range(len(path_length)):
              sum += path_length[i]
          avg = int(sum/(i+1))
          print("average path length of a path until a cycle is encountered :", avg)

          average path length of a path until a cycle is encountered : 4
```

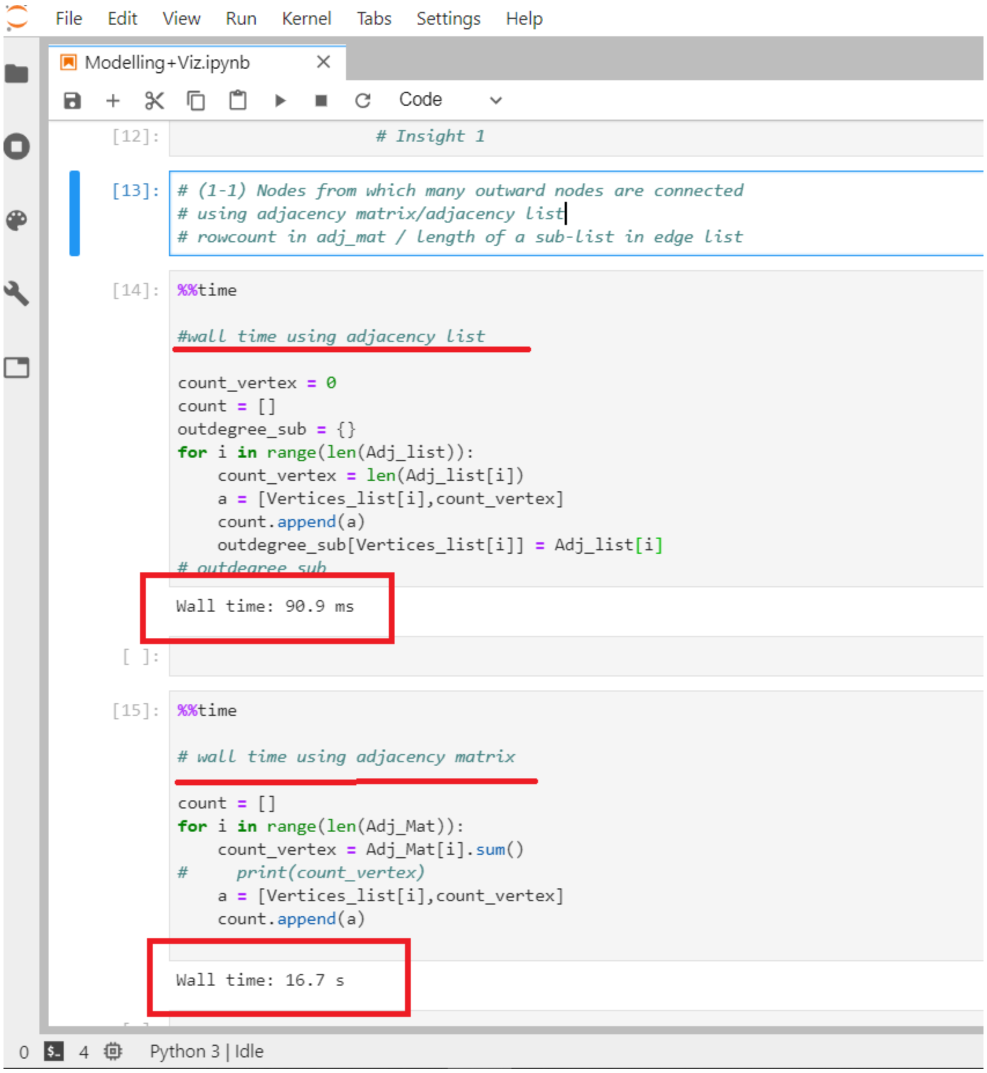
SECTION - 3

1. Scalability

Processor - intel i7 7th gen CPU 2.7GHz - 2.9GHz 64-bit OSx64 based processor
RAM - 8 gb
Tool/software - Anaconda version 2019.07, Neo4j Desktop
Language – Python 3.7

<u>Average Time taken for the execution of the above insights:</u>

Insight - [Subreddit which has the greatest number of responses (Node with the greatest number of outward edges)]

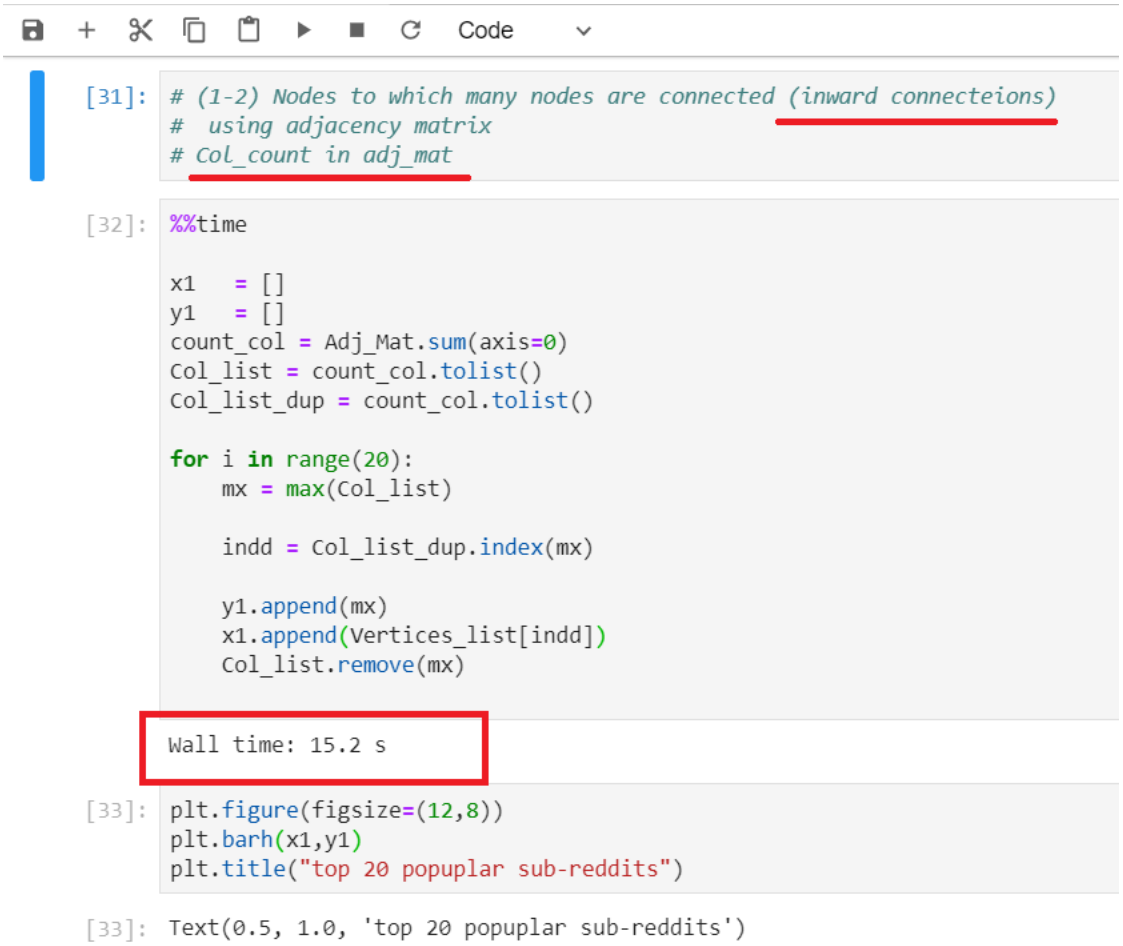


```
[12]:                      # Insight 1

[13]: # (1-1) Nodes from which many outward nodes are connected
      # using adjacency matrix/adjacency list
      # rowcount in adj_mat / length of a sub-list in edge list

[14]: %%time

      #wall time using adjacency list

      count_vertex = 0
      count = []
      outdegree_sub = {}
      for i in range(len(Adj_list)):
          count_vertex = len(Adj_list[i])
          a = [Vertices_list[i],count_vertex]
          count.append(a)
          outdegree_sub[Vertices_list[i]] = Adj_list[i]
      # outdegree_sub

      Wall time: 90.9 ms

[ ]:

[15]: %%time

      # wall time using adjacency matrix

      count = []
      for i in range(len(Adj_Mat)):
          count_vertex = Adj_Mat[i].sum()
      #     print(count_vertex)
          a = [Vertices_list[i],count_vertex]
          count.append(a)

      Wall time: 16.7 s
```

- **<u>Conclusion of the above consideration:</u>**

Analyzing the wall time shown above, we can conclude that using adjacency list is more efficient than adjacency matrix for the insight.

2. Insight - [Subreddit which answered the most of the other subreddits (Node with the greatest number of incoming edges)]



```
[31]: # (1-2) Nodes to which many nodes are connected (inward connecteions)
      #  using adjacency matrix
      # Col_count in adj_mat
```

```
[32]: %%time

      x1   = []
      y1   = []
      count_col = Adj_Mat.sum(axis=0)
      Col_list = count_col.tolist()
      Col_list_dup = count_col.tolist()

      for i in range(20):
          mx = max(Col_list)

          indd = Col_list_dup.index(mx)

          y1.append(mx)
          x1.append(Vertices_list[indd])
          Col_list.remove(mx)

      Wall time: 15.2 s
```

```
[33]: plt.figure(figsize=(12,8))
      plt.barh(x1,y1)
      plt.title("top 20 popuplar sub-reddits")

[33]: Text(0.5, 1.0, 'top 20 popuplar sub-reddits')
```

- **Conclusion of the above consideration:**
  The logic for the insight is based on column count of the adjacency matrix which comparatively takes less time than adjacency list or edge list in which we need to use 2 for loops. That’s why we considered adjacency matrix as an efficient data model for this insight.

- **Overall Conclusion of above Considerations:**
  1. Adjacency List is the best fit for the 1st Insight whereas Adjacency matrix is the best fit of the 2nd Insight.
  2. The selection of the respective data model depends upon the nature of insights.

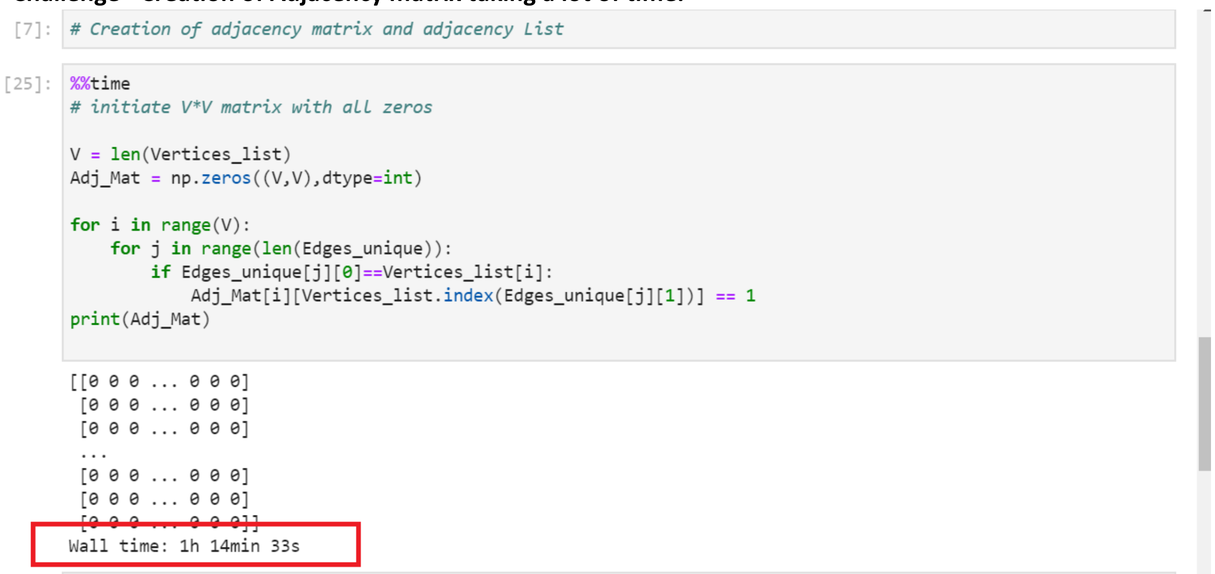
Execution time for subset of dataset for the creation of Adjacency list and matrix:

	Adjacency Matrix(wall-time)	Adjacency List(wall-time
10% of entire Vertices	1min 10 sec	1min 15sec
50% of entire Vertices	3min 40sec	3min 45sec
Entire Vertices list	7min 16sec	7min 56sec

2. Challenges and Solutions

a) Time related Challenges

1. Challenge - Creation of Adjacency matrix taking a lot of time.

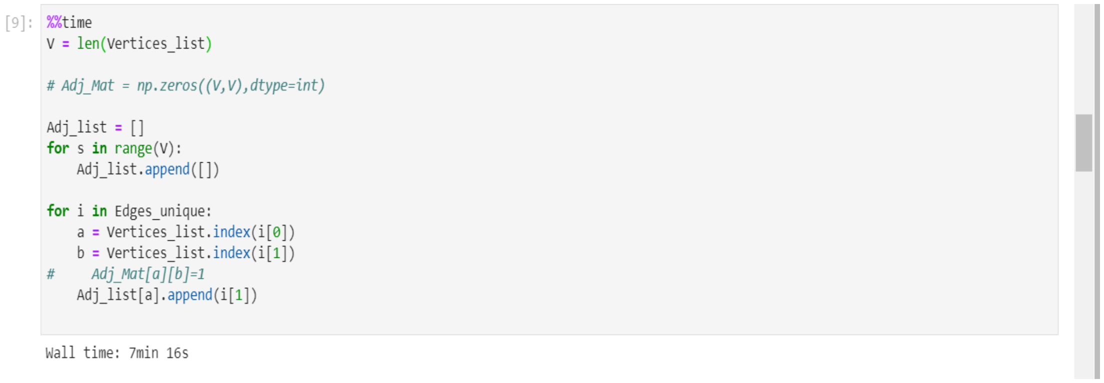


```
[7]:  # Creation of adjacency matrix and adjacency List
```

```
[25]: %%time
      # initiate V*V matrix with all zeros

      V = len(Vertices_list)
      Adj_Mat = np.zeros((V,V),dtype=int)

      for i in range(V):
          for j in range(len(Edges_unique)):
              if Edges_unique[j][0]==Vertices_list[i]:
                  Adj_Mat[i][Vertices_list.index(Edges_unique[j][1])] == 1
      print(Adj_Mat)
```

```
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
Wall time: 1h 14min 33s
```

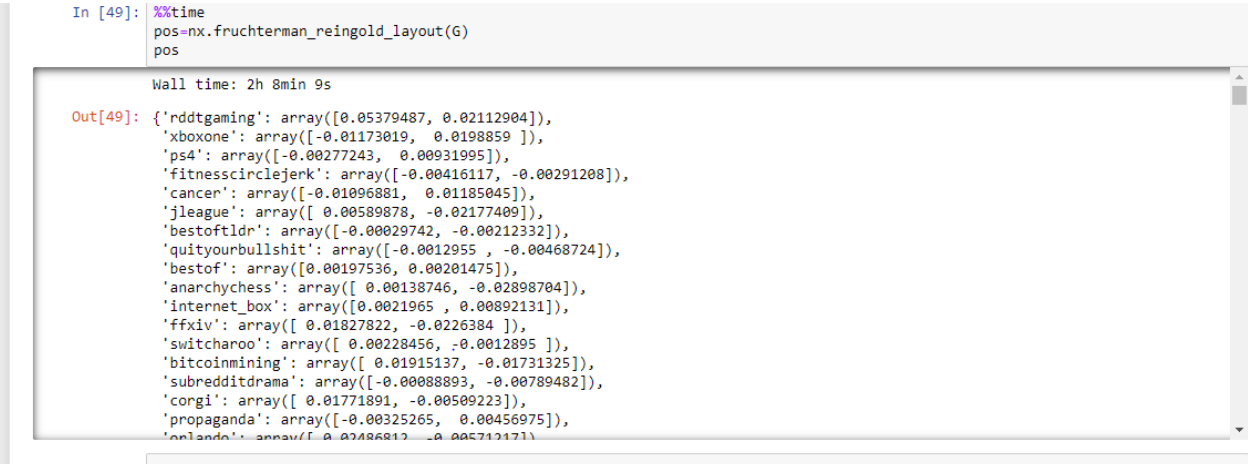
Solution – We came up with another algorithm that optimized the run time and below is the result of the optimized version.



```
[9]: %%time
     V = len(Vertices_list)

     # Adj_Mat = np.zeros((V,V),dtype=int)

     Adj_list = []
     for s in range(V):
         Adj_list.append([])

     for i in Edges_unique:
         a = Vertices_list.index(i[0])
         b = Vertices_list.index(i[1])
     #     Adj_Mat[a][b]=1
         Adj_list[a].append(i[1])
```

```
Wall time: 7min 16s
```

2. Challenge - Initially, we also considered Networkx for plotting graph structure insights, but the time for execution is much higher.

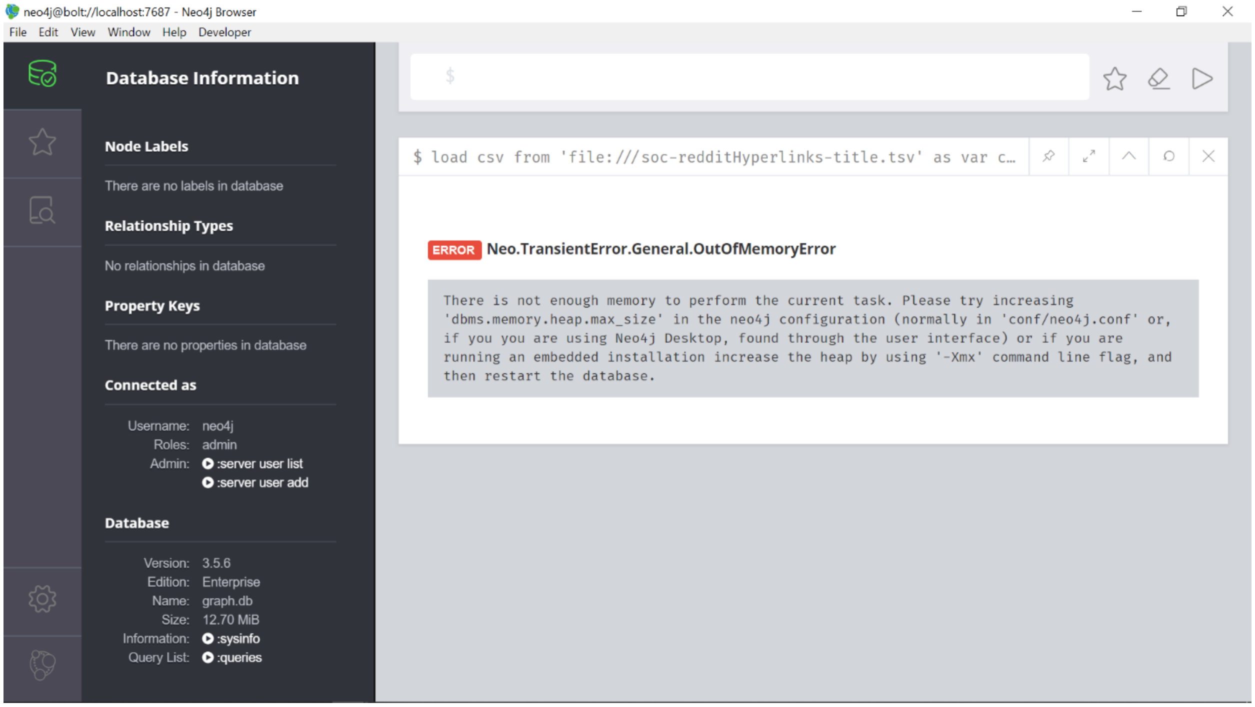


```
In [49]: %%time
         pos=nx.fruchterman_reingold_layout(G)
         pos

         Wall time: 2h 8min 9s

Out[49]: {'rddtgaming': array([0.05379487, 0.02112904]),
          'xboxone': array([-0.01173019,  0.0198859 ]),
          'ps4': array([-0.00277243,  0.00931995]),
          'fitnesscirclejerk': array([-0.00416117, -0.00291208]),
          'cancer': array([-0.01096881,  0.01185045]),
          'jleague': array([ 0.00589878, -0.02177409]),
          'bestoftldr': array([-0.00029742, -0.00212332]),
          'quityourbullshit': array([-0.0012955 , -0.00468724]),
          'bestof': array([0.00197536, 0.00201475]),
          'anarchychess': array([ 0.00138746, -0.02898704]),
          'internet_box': array([0.0021965 , 0.00892131]),
          'ffxiv': array([ 0.01827822, -0.0226384 ]),
          'switcharoo': array([ 0.00228456, -0.0012895 ]),
          'bitcoinmining': array([ 0.01915137, -0.01731325]),
          'subredditdrama': array([-0.00088893, -0.00789482]),
          'corgi': array([ 0.01771891, -0.00509223]),
          'propaganda': array([-0.00325265,  0.00456975]),
          'orlando': array([ 0.02486812, -0.00571217]),
```

Solution - We concluded not to use networks as it gives memory issues and high cpu usuage, thus it is neither space nor time efficient, finally we opted for the adjacency list, adjacency matrix instead.

b) Memory related Challenges

Below are the memory errors encountered:

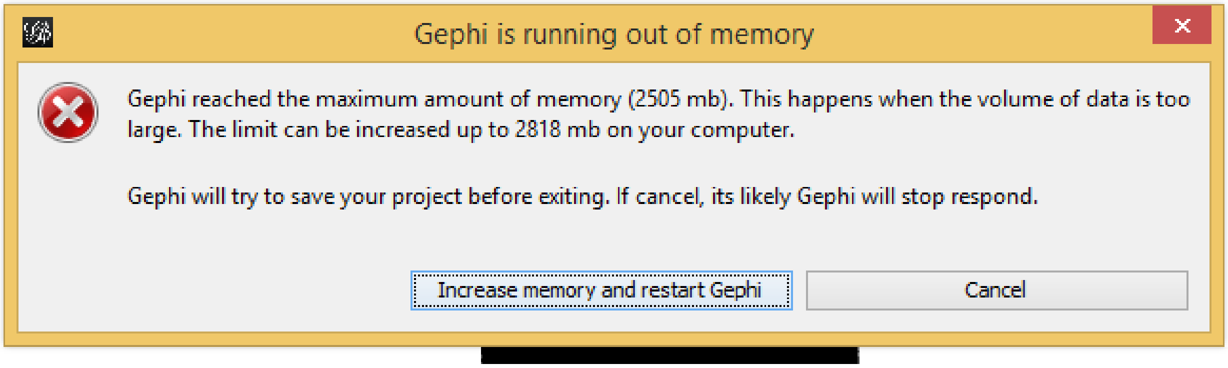


```
$ load csv from 'file:///soc-redditHyperlinks-title.tsv' as var c…
```

ERROR Neo.TransientError.General.OutOfMemoryError

```
There is not enough memory to perform the current task. Please try increasing
'dbms.memory.heap.max_size' in the neo4j configuration (normally in 'conf/neo4j.conf' or,
if you you are using Neo4j Desktop, found through the user interface) or if you are
running an embedded installation increase the heap by using '-Xmx' command line flag, and
then restart the database.
```

Solution : Increasing the heap size to 2GB from 1GB(default), solved the memory issue.



Gephi is running out of memory

Gephi reached the maximum amount of memory (2505 mb). This happens when the volume of data is too large. The limit can be increased up to 2818 mb on your computer.

Gephi will try to save your project before exiting. If cancel, its likely Gephi will stop respond.

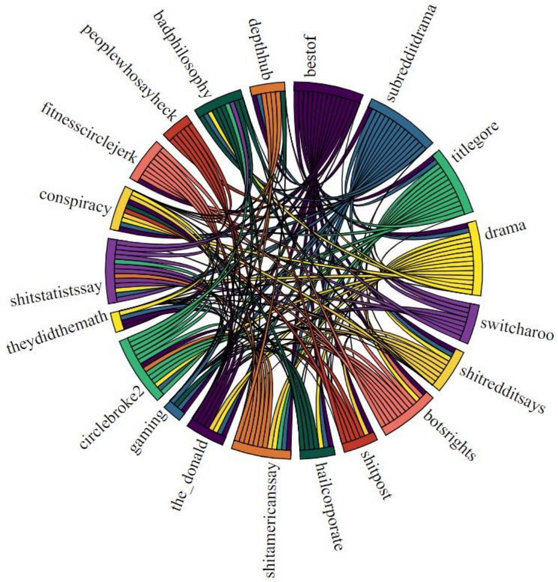
[redacted]

Conclusion: Even after increasing the memory limit to the suggested value (in the prompt), the issue was not solved for big dataset and we were unable to get insights. So, we preferred neo4j and python over gephi.

SECTION – 4

One of the given research papers speaks about Visual Analysis of Large Graphs. Chord diagram is one of the suggested visualizations for complex networks.

For the visualization of multiple components, first a layout for each individual connected component is calculated and then a specific placement of these components on the screen is performed. The most widely used placement method is called *packing*. Below is the implemented chord diagram for the given dastaset.



From research paper:
http://ezproxy.library.uvic.ca/login?url=https://doi.org/10.1111/j.1467-8659.2011.01898.x

We also researched on the usage of degree distribution in social networking by drawing inferences to the effect of influential nodes in a network and the propagation of virus/malware. We analyzed strongly connected components with respect to network analysis.

LinkedIn Paper also gives an interesting feature by working on adjacency matrix(M).
Implementing (M^2-I) gives the 2-hop nodes from a node.
http://ezproxy.library.uvic.ca/login?url=https://dl.acm.org/citation.cfm?doid=3299869.3314044