CSD 482 DATA VISUALISATION

Degree Of Separation and Visualisation on Actor-Movie Dataset

By - Group 18

Kartikeya Shrivastava 2010111055

Shashwat Tiwari 2010111038

Aman K Pendyala 2010110074

Introduction

We arbitrarily choose an actor and then connect them to another actor via a film that both actors have appeared in together, hence giving us the degree of separation between the two. We design different visualisations from the set of actors we have that give us a depiction of how they are separated from each other by the means of different movies they have played a part in and to what extent.

Softwares and Resources Used

1. Python
2. Gephi
3. Force Atlas2
4. Yifan Hu
5. Kaggle database that we worked on: https://www.kaggle.com/datasets/darinhawley/imdb-films-by-actor-for-10k-actors

Stages

---

1. For this project, we were initially planning on working on the entire IMDB actor dataset, however even simple database operations on it were taking time in the order of several minutes, so we decided to downsize to a dataset found on Kaggle that had data for the top 10,000 actors on IMDB. Since graph layout and computing network diameter are very computationally expensive operations, working on the smaller dataset itself took hours at a time at specific points. Doing so on the larger IMDB dataset would have been impossible.1

```
# Perform the merge operation
actor_actor = pd.merge(actorfilms, actorfilms, on = 'FilmID')[['Actor_x', 'ActorID_x', 'Film_x',
                                                               'Year_x', 'Rating_x', 'FilmID', 'Actor_y', 'ActorID_y']]
# Rename columns
actor_actor.rename(columns = {'Film_x': 'Film', 'Year_x': 'Year', 'Votes_x': 'Votes', 'Rating_x': 'Rating'},
                   inplace = True)
# Remove unnecessary self edges
actor_actor = actor_actor[actor_actor['Actor_x'] != actor_actor['Actor_y']]
# Reset the index
actor_actor.reset_index(drop = True, inplace = True)
# Reorder columns
actor_actor = actor_actor[['Actor_x', 'Actor_y', 'ActorID_x', 'ActorID_y', 'Film', 'Year', 'Rating', 'FilmID']]
edgelist = actor_actor.copy(deep=True)
actor_actor.tail(20)
```

2. We convert the ratings to -1 * the original rating to run algorithms like the shortest path algorithm.

```
edgelist['Rating'] = 1 / actor_actor['Rating']
edgelist.sample(10)
```

3. Change a few column names and export to CSV to input data into Gephi.

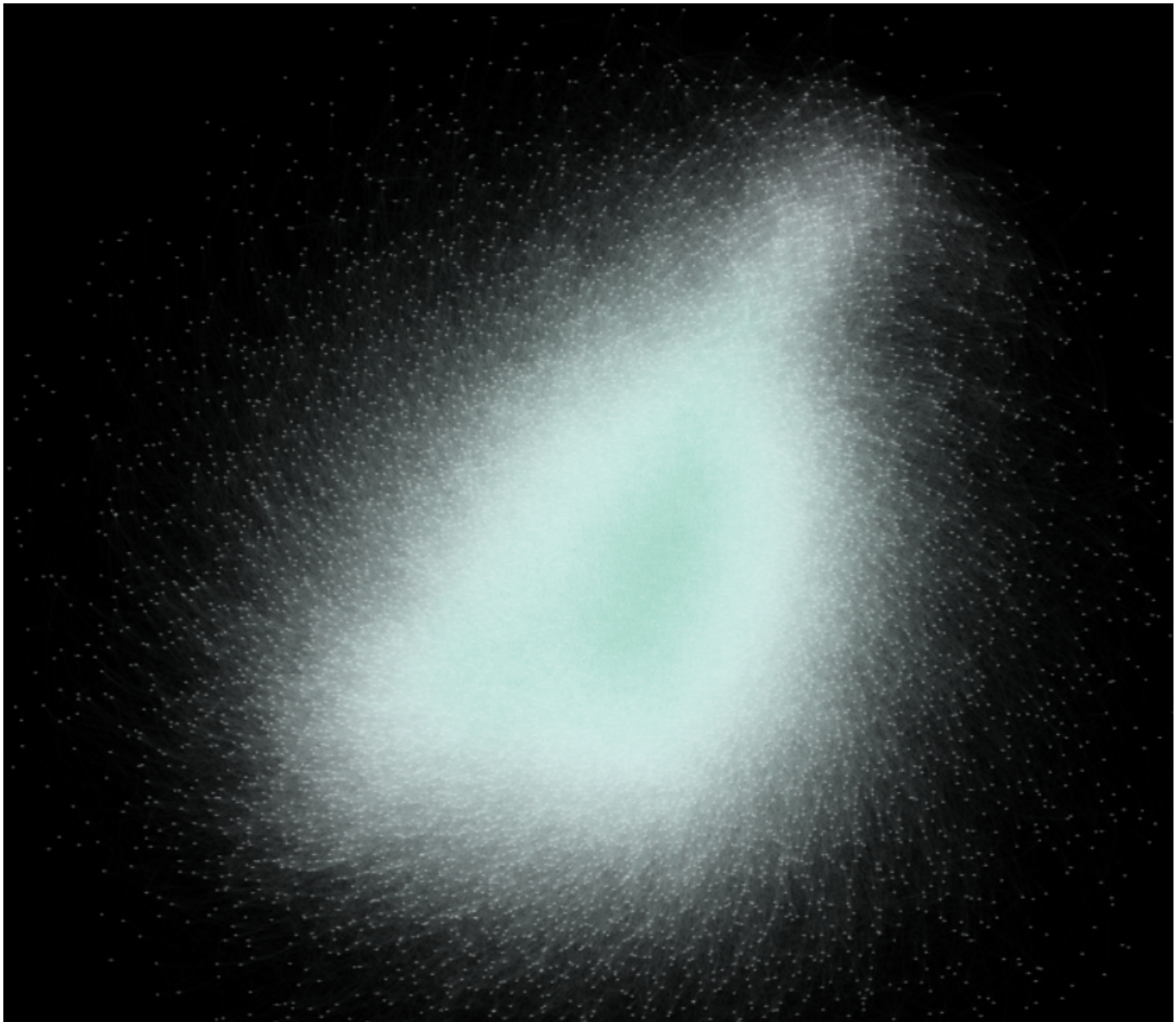
```
gephi_edges = actor_actor[['Actor_x', 'Actor_y']].copy(deep = True)
gephi_edges.rename(columns = {'Actor_x': 'Source', 'Actor_y': 'Target'}, inplace = True)
gephi_edges.head(10)
```

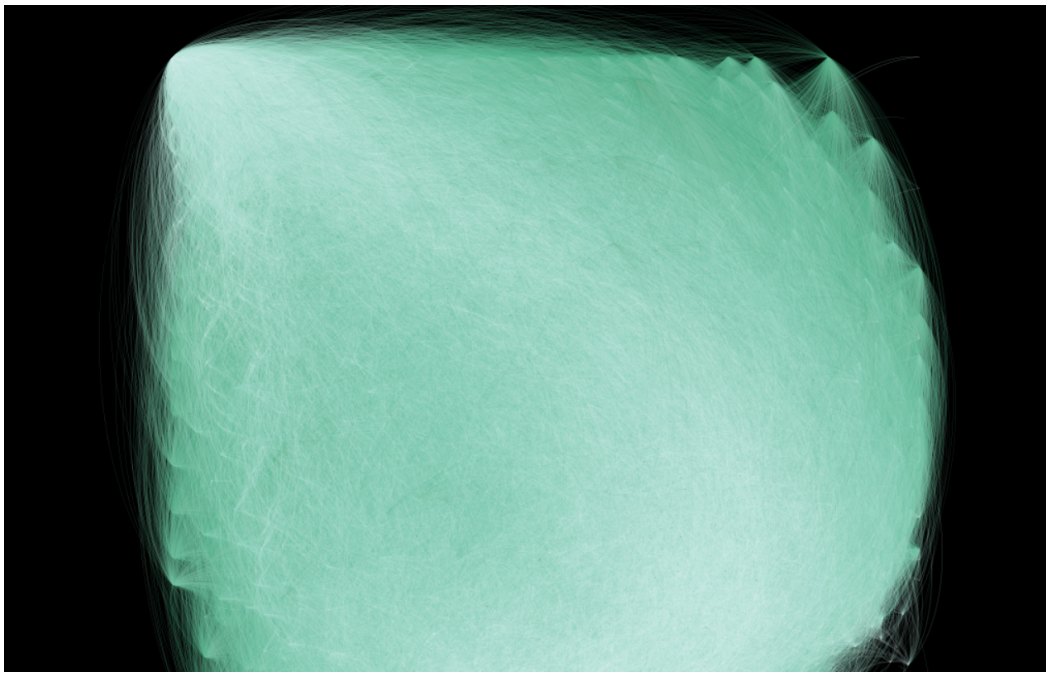
4. Load the data into a network multigraph with the (1 / Ratings) as edge weights and the film's name as the key of an edge. We set the edge ratings as the reciprocal of the ratings to run the shortest paths and MST algorithms later on.

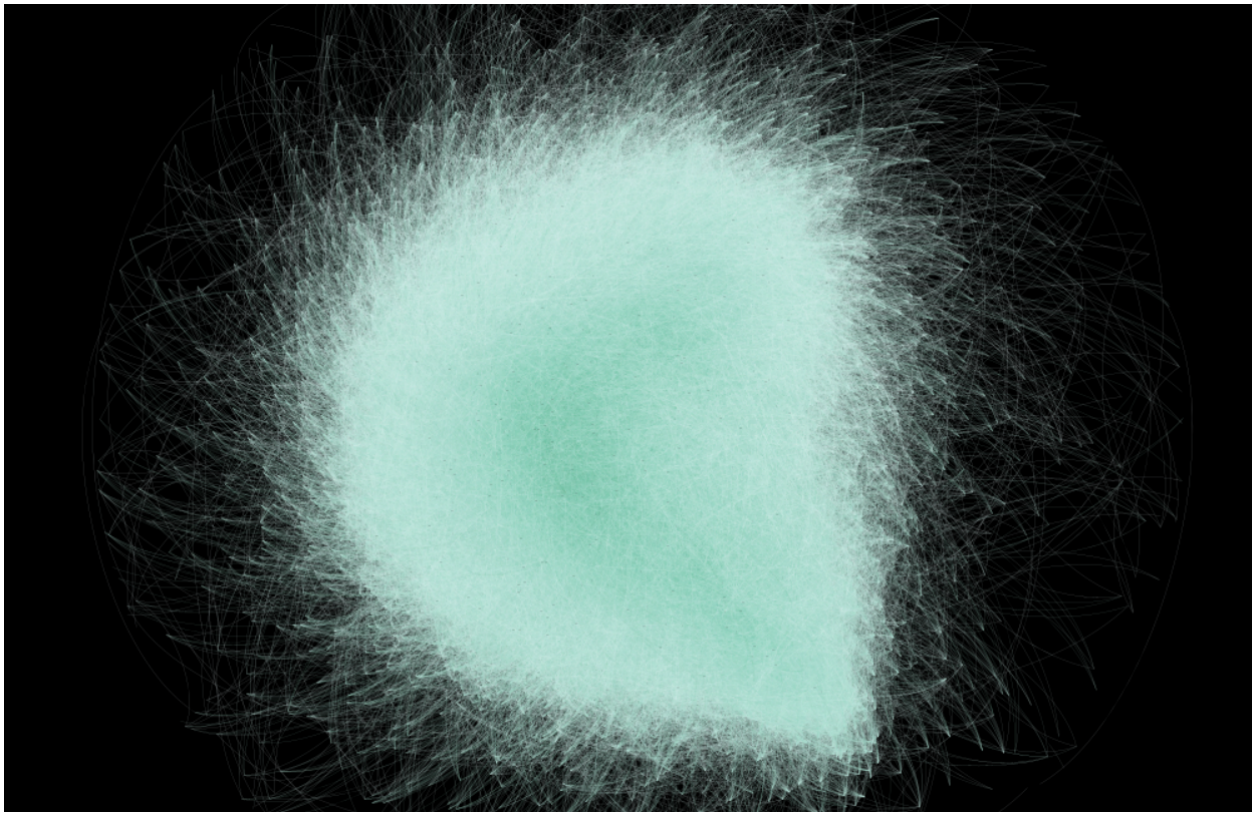
```
# Create a multigraph
G = nx.from_pandas_edgelist(edgelist, source = 'Actor_x', target = 'Actor_y',
                            edge_attr = ['Film', 'Rating'], create_using = nx.MultiGraph())
# and a simple graph
simple_G = nx.from_pandas_edgelist(edgelist, source = 'Actor_x', target = 'Actor_y', create_using = nx.Graph())
```

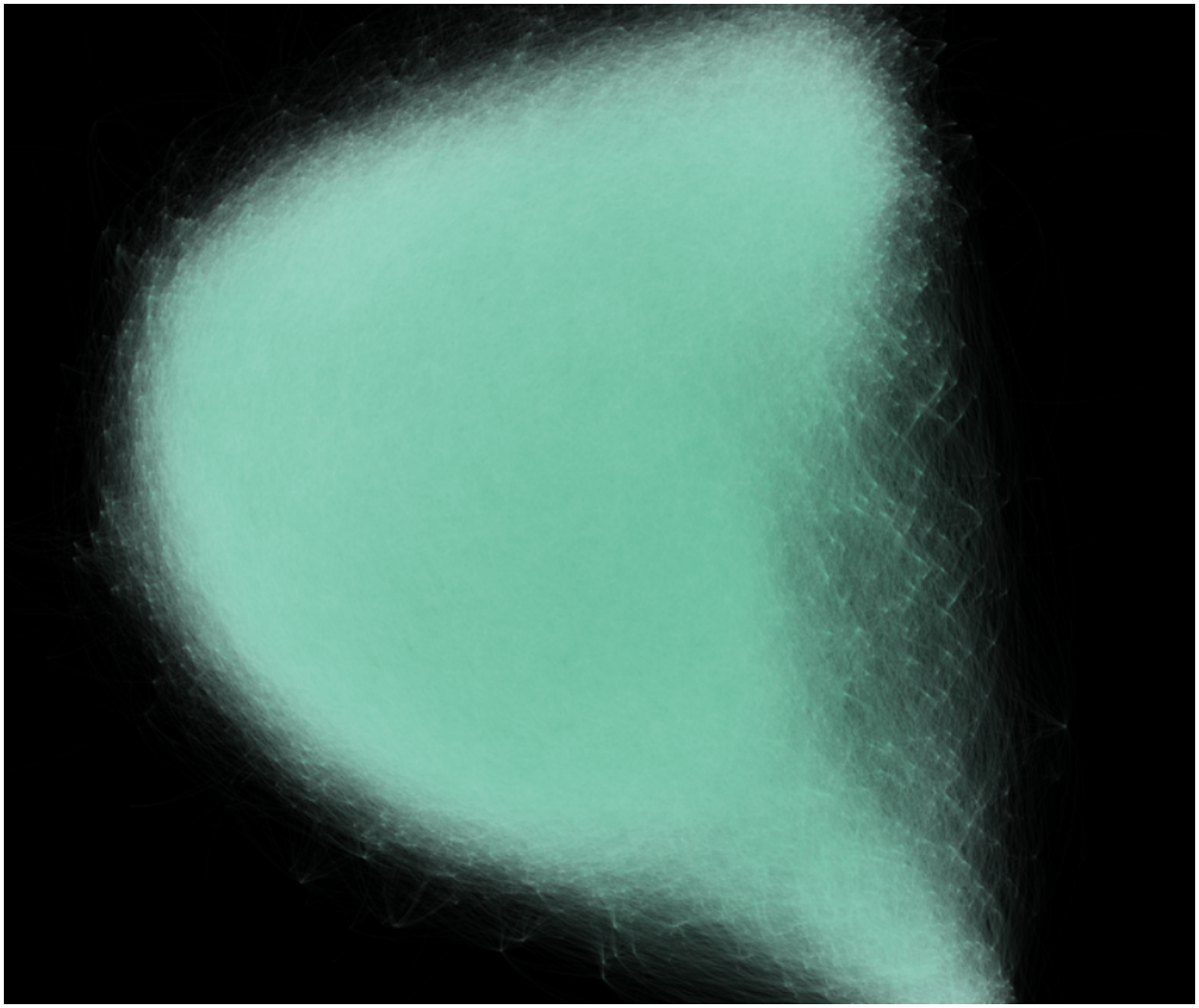
Moving on to the Data Visualisation and Analysis Part

1. We now input this data into Gephi to get an initial overview of the structure of the relationship presents here. Running a layout algorithm on a massive graph with nearly 10,000 nodes and over 600,000 edges proves very computationally expensive. We ran this for upwards of 2 hours to get a 15,000 x 15,000 pixel 200MB image.









Drawbacks with the current model -

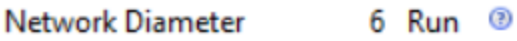
The issue we faced while creating the above-displayed visualisations was their complexity. Even for a data set of merely 10,000 actors, we had reached a point where the number of edges between the actors would total up to 600,000.

Even though these visualisations are appealing and look aesthetically pleasing they aren't very informative, which was the whole point of our project, to begin with.

Therefore, we move on to some other means to analyse the data set which are more discernable.

The second approach - Diameter and Degree of Separation

1. Using Gephi to analyse the network diameter of the given graph, we get



2. This is like the standard social networks idea of 'Six Degrees of Separation wherein any two people can be connected in a minimum number of six steps, the popular film version of this is 'Six Degrees of Kevin Bacon' where any actor can be connected to Kevin Bacon within 6 movies.

(Six Degrees of Separation: https://en.wikipedia.org/wiki/Six_degrees_of_separation)

(Six Degrees of Kevin Bacon: https://en.wikipedia.org/wiki/Six_Degrees_of_Kevin_Bacon)

3. Connected Components:

We first try to understand the number of connected components in the graph; since this is a graph of the top 10,000 actors on IMDB, preliminary intuition would suggest that we might just get one connected component since famous actors tend to work a lot with others.

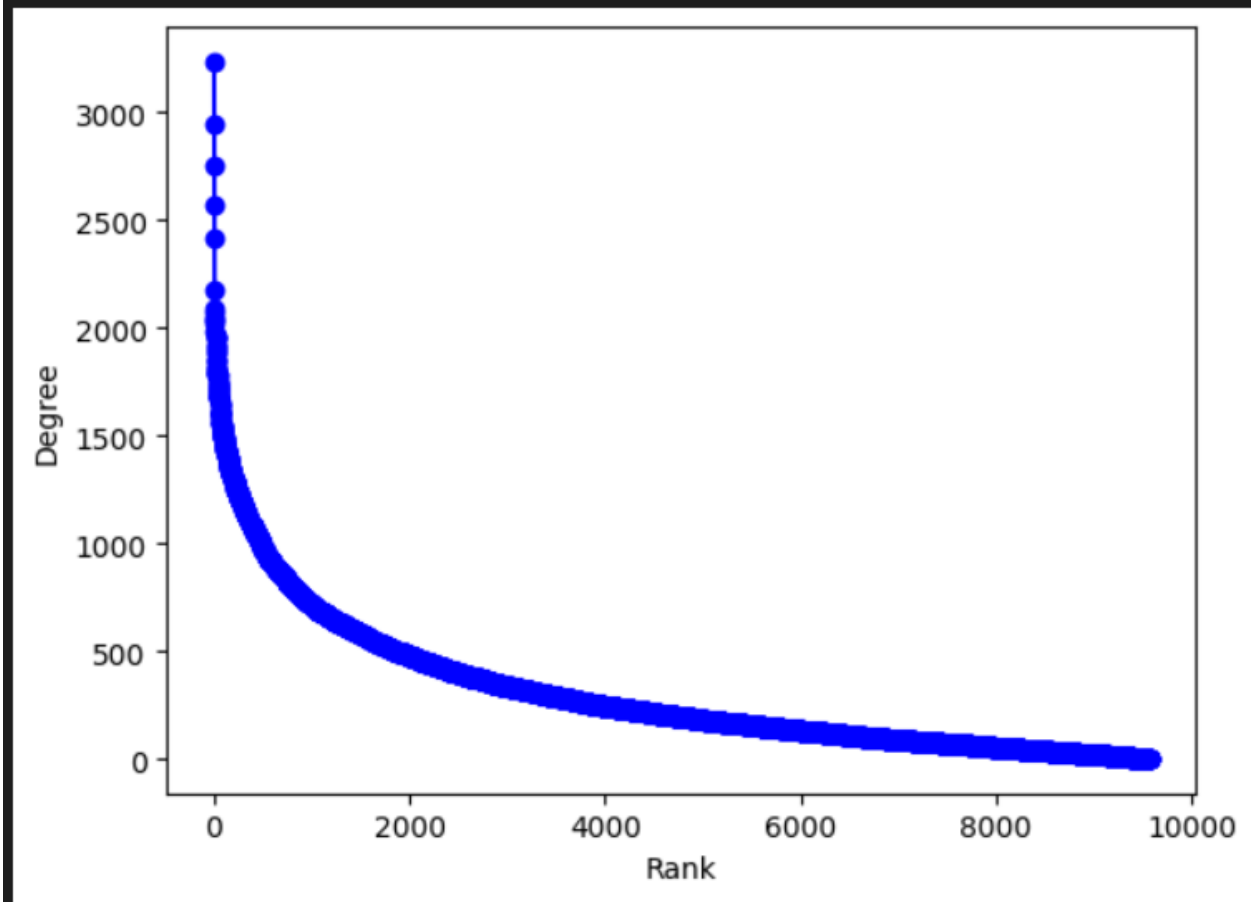
As observed, we're getting one substantial connected component consisting of nearly all the nodes, and 2 tiny components.

This resembles a typical social network phenomenon of 'Giant Components' wherein in any sufficiently sized graph, one connected component will encompass a significant fraction of the entire graph's vertices.

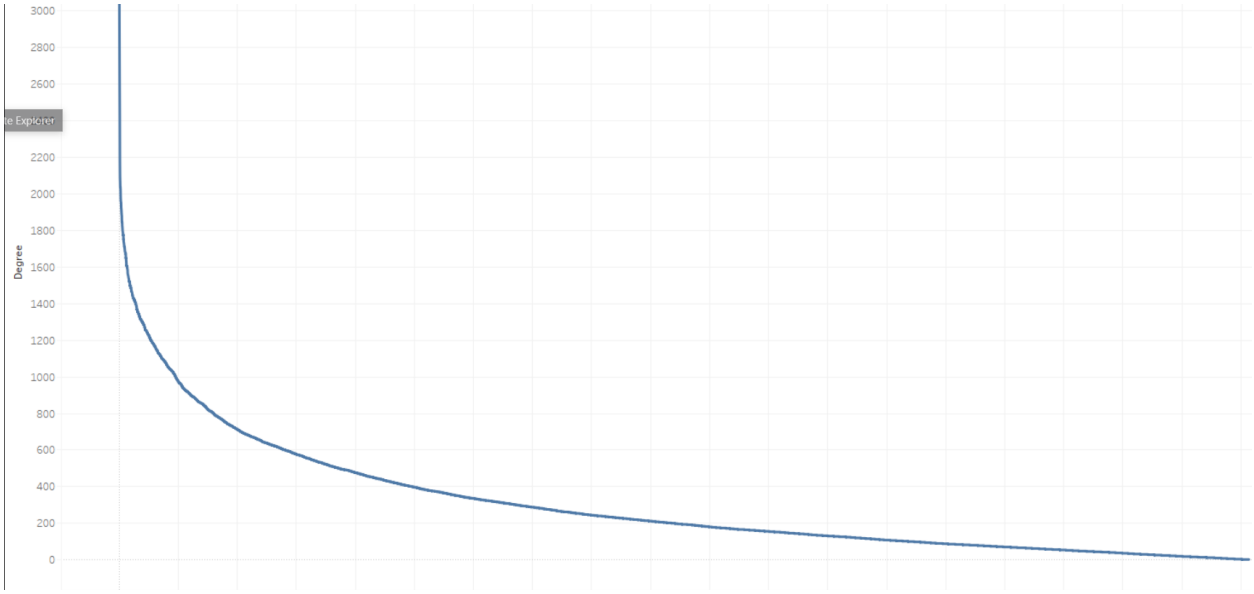
(https://en.wikipedia.org/wiki/Giant_component)

4. Degree Analysis

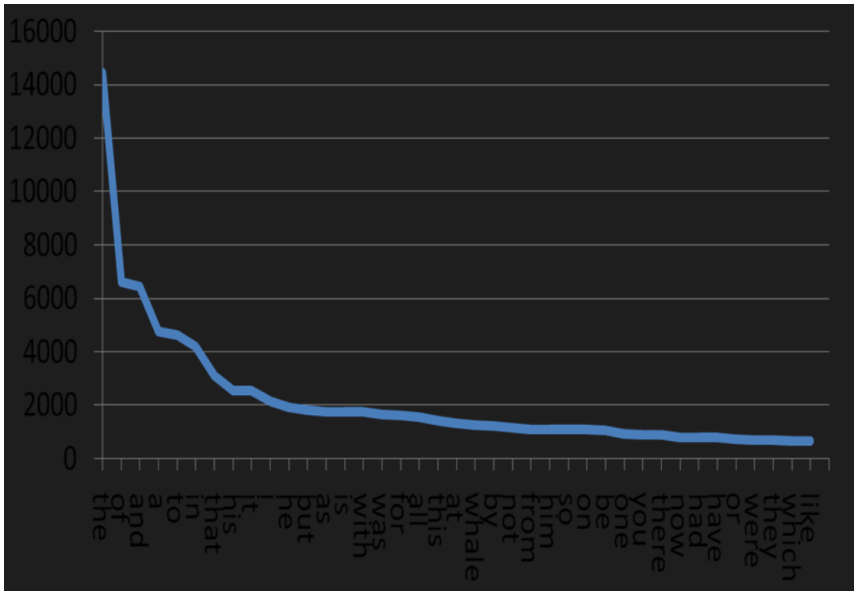
```
degree_sequence = sorted((d for n, d in G.degree()), reverse=True)
plt.plot(degree_sequence, "b-", marker="o")
plt.ylabel("Degree")
plt.xlabel("Rank")
plt.show()
```



Exporting this into tableau we get the following visual:-



Interestingly, the graph above, plotting the degrees w.r.t to rank of the particular degree, seems to mimic the famous statistics law called 'Zipf's law' where in the rank and frequency distribution is often observed to be an inverse relation, a popular rendition of this law is in the frequency of words in the English language, where the 2nd most used word has half the frequency of the most used word, the third most used word has one third the frequency, and so on.



Best attempt at a discernable visualisation (yet)

Minimum Spanning Tree

1. First, we create the minimum spanning tree on the largest component of graph

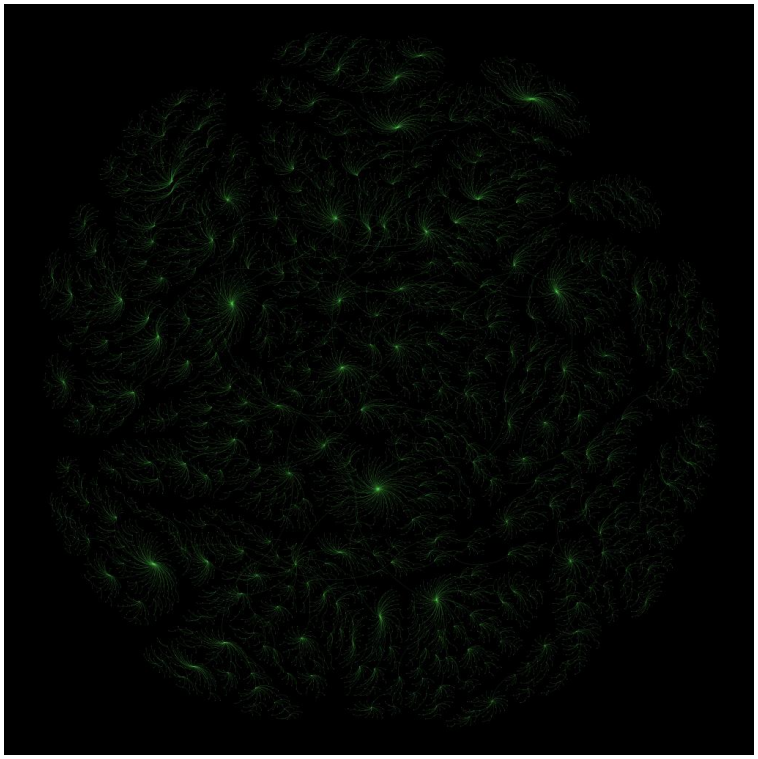
2. Then export it to Gephi to create graphs.

3. The minimum spanning tree of a graph is a subset of edges of a connected weighted graph that connects all the vertices without any cycles, and is of minimum possible weight.

4. The use of MST here could portray actor relations but with edges portraying the best movies created among the actors, representing the finest works produced by the movie industry; graphing this information as a network lets you observe how actors fit in with these movies.

(Minimum Spanning Tree: https://en.wikipedia.org/wiki/Minimum_spanning_tree)

5. Porting this into Gephi and graphing using Force Atlas 2 + Yifan Hu, we get:



Benefits of the Minimum Spanning Tree

1. Still complex looking, however, one can observe the formation of loops and different substructures making it a bit simpler to discern relations.
2. Labels can be added to the minimum spanning tree making it even more legible and clear in terms of representing the data set.
3. Since its an MST the number of edges is also now brought down to 10,000 same as the number of actors (nodes) in the data set, earlier this number was around 600,000.

Added Feature - Since we negated the rating at the very beginning of our project, one added functionality this MST serves is that it connects the two actors in question with their most popular movies.

This, in earlier visualisations, was an issue as certain superstars had multiple mega-hit movies. Hence, they clustered up around the centre and made the visualisations hard to understand.

Shortest Path

```
# Returns the shortest path from a source actor to a target actor, along the path of the highest rated movies in between
def shortestpath(source, target):
    shortest_path = nx.shortest_path(G, source = source, target = target, weight = 'Rating')
    movie_list = []
    for i in range(1, len(shortest_path)):
        currdf = actor_actor[actor_actor['Actor_x'] == shortest_path[i - 1]]
        currdf = currdf[currdf['Actor_y'] == shortest_path[i]]
        movie_list.append(actor_actor.iloc[currdf['Rating'].idxmax()]['Film'])

    for i in range(0, len(shortest_path) - 1):
        print(shortest_path[i], "was in", movie_list[i], "with", shortest_path[i + 1])
```

```
shortestpath('Salman Khan', 'Brad Pitt')
```

```
Salman Khan was in Kuch Kuch Hota Hai with Anupam Kher
Anupam Kher was in Silver Linings Playbook with Robert De Niro
Robert De Niro was in Sleepers with Brad Pitt
```

The above code is a proof of concept for a simple shortest path implementation similar to the one found in Oracle of Bacon, which returns the shortest path between two actors and the movies that connect the two actors. The unique thing I tried to add with this

implementation compared to Oracle of Bacon was to make sure that the path between the two actors would go through the highest-rated movies in between them.

(Oracle of Bacon: https://oracleofbacon.org/)

Future Scope

We aim to make more pleasing visualisations to represent our data set and along the way try to increase the size of the data set as well.

Our goal is also to be able to make discernable visualisations which are not only aesthetically pleasing as well informative.

Dolor sit amet

Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer possim assum. Typi non habent claritatem insitam; est usus legentis in iis qui facit eorum claritatem. Investigationes demonstraverunt lectores legere me lius quod ii legunt saepius. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan.