Finding Key Influential Actor in Movie Industry

Potchara Vinitwattanakoon 6210412003
Department of Applied Statistics
National Institue of Development Administration
potchara.vin@stu.nida.ac.th

Ativit Chaninchodeuk 6220412019
Department of Applied Statistics
National Institue of Development Administration
ativit.chan@stu.nida.ac.th

บทคัดย่อ : ในวงการภาพยนตร์นักแสดงมีส่วนสำคัญมากในการสร้างหนังให้ประสบความสำเร็จ ในการคัดเลือกนักแสดงนอกจากจะต้องดูว่านักแสดงแต่ละคนมีความสามารถในการเล่นหนังประเภทนั้นแล้ว จะต้องดูในเรื่องของความสามารถในการแสดงหนังร่วมกับนักแสดงผู้อื่นอีกด้วย งานวิจัยนี้จะทำการวิเคราะห์เครือข่ายของนักแสดงหนังเพื่อหาความสัมพันธ์ของนักแสดงแต่ละคนที่เคยแสดงหนังร่วมกัน (Co-Starring) และหานักแสดงที่จุดศูนย์กลางของเครือข่ายการแสดงหนังร่วมกันโดยดูจากค่า centrality ที่ได้จาก network graph จากนั้นจึงทำการหา community ภายในกลุ่มของนักแสดงหนัง

คำสำคัญ : Network Graph, Community Detection, Centrality, Louvain algorithm, Movie, Actor

1. บทนำ

งานวิจัยชิ้นนี้มีเป้าหมายคือการหานักแสดงที่เป็นจุดศูนย์กลางในวงการหนังโดยใช้การวิเคราะห์เครือข่ายความสัมพันธ์ของการแสดงหนังร่วม (Co-Starring) ที่แสดงในรูปแบบของกราฟเครือข่าย (Network Graph) ที่มีจุดเป็นนักแสดง เพื่อหาค่า centrality ของจุดนักแสดงแต่ละจุด จากนั้นจึงทำการหา community ที่มีอยู่ในกราฟเครือข่ายด้วย community detection algorithm และดูลักษณะการกระจายตัวของนักแสดงแต่ละคน

การหาจุดนักแสดงที่เป็นจุดศูนย์กลางในวงการหนังนอกจากจะพบว่านักแสดงคนใดที่เป็นจุดศูนย์กลางของเครือข่ายการแสดงหนังร่วมแล้ว เป็นการทำให้ทราบว่านักแสดงคนใดเป็นนักแสดงที่มีอิทธิพลในวงการหนัง ซึ่งจะเป็นประโยชน์กับบุคคลภายนอกที่ไม่ได้อยู่ในวงการหนังได้รู้จักนักแสดงมากขึ้นโดยใช้การวิเคราะห์กราฟเครือข่าย

2. ทฤษฎีและงานที่เกี่ยวข้อง

กราฟ (Graph) เป็นโครงสร้างที่ใช้แทนความสัมพันธ์ของข้อมูลประเภทหนึ่งอันประกอบด้วยเซตของจุด (Node or Vertex) มากกว่า 2 จุดขึ้นไป ซึ่งเชื่อมต่อกันด้วยเส้นเชื่อม (Edge) ที่แสดงถึงความสัมพันธ์ระหว่างจุดสองจุด กราฟที่เกิดขึ้นสามารถแบ่งได้ออกได้เป็น 2 ประเภทตามลักษณะความสัมพันธ์ระหว่างจุดสองจุด คือ กราฟแบบมีทิศทาง และกราฟแบบไม่มีทิศทาง

กราฟแบบมีทิศทาง (Directed Graph) คือ กราฟที่มีเส้นเชื่อมระหว่างจุดสองจุดที่มีหัวลูกศรชี้แสดงทิศทางความสัมพันธ์ของจุดทั้งสองนั้น ยกตัวอย่างเช่น กำหนดให้จุดแทนบุคคล และเส้นเชื่อมแทนทิศทางการเคลื่อนที่ของข้อมูลข่าวสารบ่งบอกว่าข้อมูลข่าวสารนั้นเดินทางจากใครไปหาใคร เป็นต้น โดยทิศทางของความสัมพันธ์จะต้องเป็นไปในทิศทางเดียวกันกับที่หัวลูกศรชี้เท่านั้น

กราฟแบบไม่มีทิศทาง (Undirected Graph) คือ กราฟที่มีเส้นเชื่อมระหว่างจุดสองจุดที่ไม่มีหัวลูกศร บ่งบอกถึงความสัมพันธ์ระหว่างจุดสองจุดนั้นซึ่งที่มีค่าเท่ากัน หรือทิศทางการเคลื่อนที่ได้ทั้งสองทาง เป็นต้น

น้ำหนัก (Weight) ของเส้นเชื่อมจะเป็นตัวบ่งบอกน้ำหนักความสัมพันธ์ระหว่างจุดสองจุด ซึ่งอาจมีหรือไม่มีก็ได้ หากไม่มีน้ำหนักในเส้นเชื่อมทุกเส้นจะแสดงว่ามีน้ำหนักเท่ากันหมดทุกเส้นเชื่อมคือหนึ่งหน่วย น้ำหนักเส้นเชื่อมจะมีประโยชน์อย่างมากในการประยุกต์ใช้งานและบ่งบอกถึงความสัมพันธ์ของเส้นเชื่อมนั้นว่ามีมากน้อยแค่ไหน

Centrality ใช้สำหรับการวัดค่าความเป็นจุดศูนย์กลางของสมาชิกในเครือข่ายของแต่ละจุด ถูกคิดค้นขึ้นโดย Freeman (1979) ซึ่งประกอบด้วย degree centrality, closeness centrality, betweenness centrality, และ eigenvector centrality (Yan; & Ding. 2009)

Degree Centrality ใช้วัดว่าสมาชิกใดบ้างที่เป็นจุดศูนย์กลางของการเชื่อมโยง ซึ่งเป็นตำแหน่งที่มีอิทธิพลสูงสุดในเครือข่าย วัดได้จากจำนวนเส้นเชื่อมโยงทั้งหมด ที่โยงมาจากสมาชิกเครือข่ายอื่น ๆ ทั้งที่อยู่ภายในกลุ่มเดียวกันและตรงข้ามกลุ่มสามารถคำนวณได้จากสมการ

$$C_D(v) = \frac{d_v}{|N|-1}$$

โดยที่ N คือเซตของจุด
d_v คือ ดีกรีของจุด v

Closeness Centrality ใช้วัดหาค่าความเป็นจุดศูนย์กลางของจุดในเครือข่าย โดยวัดจากผลรวมระยะทางของเส้นทางที่สั้นที่สุด (Shortest Paths) ระหว่างจุดที่สนใจกับจุดทุกจุดทั้งหมดที่เหลือในเครือข่าย เป็นค่าที่คำนวณโดยคิดจากระยะทาง Euclidean Distance ของเส้นทางที่สั้นที่สุดที่เชื่อมระหว่างจุดที่สนใจกับจุดอื่นทุกจุดภายในเครือข่ายทั้งหมด สามารถคำนวณได้จากสมการ

$$C(u) = \frac{n-1}{\sum_{v=1}^{n-1} d(u,v)}$$

โดยที่ $d(v,u)$ คือ ระยะทางที่สั้นที่สุดระหว่าง v,u

Betweenness Centrality ใช้วัดว่าสมาชิกจุดใดบ้างที่มีจุดตำแหน่งเป็นตัวกลางที่เป็นทางผ่านของเส้นทางที่สั้นที่สุดในการเชื่อมสมาชิกคู่อื่นทั้งหมดในกราฟเข้าด้วยกัน ใช้วิธีคำนวณจากสัดส่วนของเส้นทางระยะทางที่สั้นที่สุด (Shortest Paths) ในการเชื่อมโยงระหว่างสมาชิกแต่ละคู่อันดับ สมาชิกที่มีค่า Betweenness Centrality สูงจะบ่งบอกว่าสมาชิกนั้นเป็นจุดทางผ่านที่อยู่ในเส้นทางระยะทางที่สั้นที่สุดระหว่างคู่สมาชิกอื่นบ่อย เป็นเหมือนสะพานทางผ่านที่เชื่อมคู่จุดสมาชิกอื่นในกราฟเครือข่ายบ่อย หรือมีความเป็นจุดศูนย์กลางมากเพราะในเส้นทางที่สั้นที่สุดของคู่สมาชิกอื่นในกราฟนั้น เส้นทางส่วนใหญ่ต้องผ่านสมาชิกจุดที่สนใจซึ่งเป็นจุดศูนย์กลางของกราฟเครือข่าย สามารถคำนวณได้จากสมการ

$$C_B(v) = \sum_{s,t \in V} \frac{\sigma(s,t|v)}{\sigma(s,t)}$$

โดยที่ V คือ เซตของจุด
$\sigma(s,t)$ คือ จำนวนระยะทางที่สั้นที่สุดระหว่าง s,t
$\sigma(s,t|v)$ คือ จำนวนเส้นทางที่ผ่านจุด v

Eigenvector Centrality ใช้วัดว่าสมาชิกตำแหน่งใดเป็นผู้มีอิทธิพลมากที่สุดในเครือข่าย โดยการนิยามว่าเพื่อนบ้านแต่ละคนมีความสำคัญไม่เท่ากัน ความสำคัญของจุดในเครือข่ายจะเพิ่มขึ้นจาก การเชื่อมต่อกับจุดยอดอื่น ๆ ที่มีความสำคัญ สามารถหาคำนวณได้จากสมการ

$$Ax = \lambda x$$

โดยที่ A คือ Adjacency Matrix ของกราฟ G
λ คือ Eigen Value

Clustering Coefficient ใช้วัดเพื่อหาค่าว่าจุดเพื่อนบ้าน (Neighbor node) ของจุดที่สนใจมีการเชื่อมต่อระหว่างเพื่อนบ้านกันเองดีแค่ไหนสามารถคำนวณได้จากสมการ

$$C_i = \frac{2e_i}{k_i(k_i-1)}$$

โดยที่ e_i คือจำนวนเส้นเชื่อมระหว่างเพื่อนบ้านกันเองของ node i
k_i คือจำนวนดีกรีของจุด i

Louvain Algorithm เป็นเครื่องมือที่ใช้ในการหา community ถูกพัฒนาขึ้นโดย Blondel et al (2008) เป็น algorithm ที่สามารถหากลุ่มของ network ที่มีขนาดใหญ่ ซึ่งในการทำงานภายในของ Louvain algorithm นั้นจะคำนวณโดยอาศัยการเปรียบเทียบค่า modularity (Q) ที่ใช้วัดค่าความเป็น community ก่อนและหลังการรวมจุดสองจุดให้เป็น community เดียวกัน เพื่อใช้ในการ cluster community ภายในกราฟเครือข่าย ในขั้นตอนการทำงานจะแบ่งออกเป็น 2 ขั้นตอน

ขั้นตอนที่ 1: กำหนดให้แต่ละจุดที่อยู่ในกราฟเป็น community ที่แตกต่างกัน (1 จุดเท่ากับ 1 community) หลังจากนั้นเลือกจุด i ใน community ที่มีเพื่อนบ้าน j และย้าย i ไปอยู่ใน community j เพื่อดูค่าค่า modularity ถ้าค่าเพิ่มขึ้นมากสุดจะรวม 2 จุดนั้นเข้าด้วยกันเป็น community เดียวกัน สามารถคำนวณได้จากสูตร

$$\Delta Q(i \to C) = \left[\frac{\sum_{in} + k_{i,in}}{2m} - \left(\frac{\sum_{tot} + k_i}{2m}\right)^2\right] - \left[\frac{\sum in}{2m} - \left(\frac{\sum tot}{2m}\right)^2 - \left(\frac{k_i}{2m}\right)^2\right]$$

โดยที่ $\sum in$ คือ ผลรวมน้ำหนักระหว่างจุดใน C
$\sum tot$ คือ ผลรวมน้ำหนักของทั้งหมดทุกจุดใน C
k_i, in คือ ผลรวมน้ำหนักระหว่างจุด i กับจุด C
k_i คือ ผลรวมน้ำหนักทั้งหมดของจุด i

จากสูตรข้างต้นเป็นการย้ายจุด i ไปยัง community C เท่านั้น เราต้องคำนวณ $\Delta Q(D \to i)$ คือการคำนวณจุด i ย้ายออกจาก community D ด้วยจากนั้นจึงสามารถคำนวณหา ΔQ ได้จากสมการ

$$\Delta Q = \Delta Q(i \to C) + \Delta Q(D \to i)$$

ขั้นตอนที่ 2: จาก community ที่ได้จากขั้นตอนที่ 1 รวมเป็นแต่ละ super node แล้ว หลังจากนั้นให้คำนวณหาค่าน้ำหนักระหว่าง 2 super node นั้นโดยใช้ผลรวมของน้ำหนักของเส้นเชื่อมที่เชื่อมระหว่าง 2 super node จากนั้นจึงวนกลับไปสู่ขั้นตอนที่ 1 เพื่อหาค่า modularity ที่เปลี่ยนแปลงอีกครั้ง และจะวนระหว่างสองขั้นตอนจนกว่าจะไม่สามารถทำให้ค่า modularity สูงไปกว่านี้ได้ จึงจะได้ community ที่อยู่ในกราฟเครือข่ายนั้น

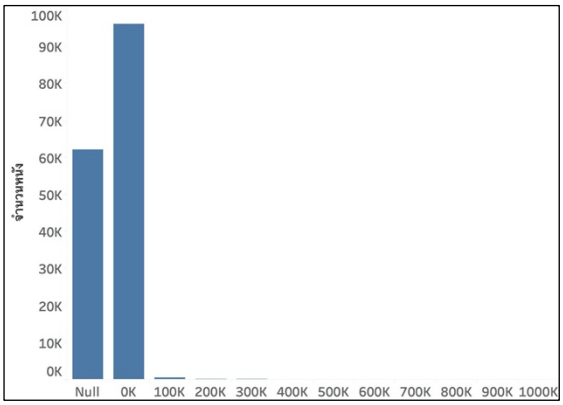
3. ข้อมูล

ที่มาของข้อมูลนั้นมาจาก The Movie Database (TMDb) โดยอาศัยการดึงข้อมูล 20 ปีย้อนหลัง ตั้งแต่ปี 2001 - 2020 ผ่าน API และสร้างออกมาเป็นตาราง dataset โดยตารางที่จะใช้ในการวิเคราะห์จะประกอบด้วย 2 ตารางคือ 1. movie.csv ซึ่งแต่ละแถวมีข้อมูลของหนังแต่ละเรื่อง 2. crew.csv ซึ่งแต่ละแถวแสดงข้อมูลของหนังและนักแสดง

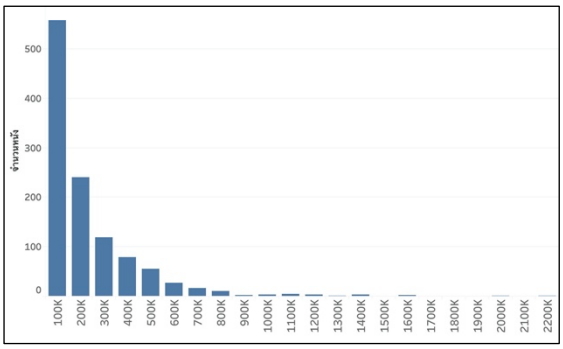
จากนั้นจึงทำการสร้างกราฟเครือข่ายขึ้นมา โดยมีจุดแต่ละจุดแทนด้วยนักแสดงแต่ละคน มีเส้นเชื่อมระหว่างสองจุดแทนการเคยแสดงร่วมกัน (Co-Starring) ระหว่างจุดนักแสดงทั้งสองจุด มีน้ำหนักของแต่ละเส้นเชื่อมตามจำนวนครั้งของหนังที่นักแสดงทั้งสองคนแสดงร่วมกัน

Data Preprocessing

สำหรับ dataset movie.csv ที่มีอยู่นั้นเป็น dataset ของข้อมูลหนังทั้งหมดที่เคยมีการจดบันทึกไว้ ซึ่งมีขนาดใหญ่มาก หากดูจำนวนของหนังแบ่งตาม bins ของ vote count ที่มีขนาด 100,000 vote count จะพบว่ามีหนังที่มี vote count ต่ำ ๆ ที่ไม่ค่อยมีคนรู้จักเป็นจำนวนมาก ทำให้ต้องมีการคัดข้อมูลหนังออกเพื่อลดจำนวนหนังลง โดยเริ่มจากการเลือกหนังที่มี vote count มากกว่า 100,000 เป็นต้นไปก่อนซึ่งเป็นหนังที่น่าจะเป็นที่รู้จักของกลุ่มคนทั่วไป



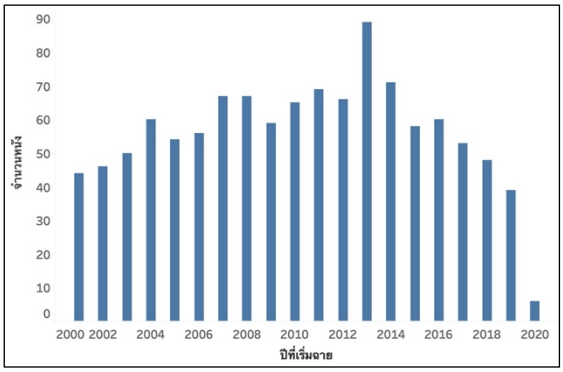
รูปที่ 1 จำนวนของหนังแบ่งตามขนาด vote count ก่อน filter



รูปที่ 2 จำนวนของหนังแบ่งตามขนาด vote count หลัง filter

ถัดไปจะเป็นการคัดกรองข้อมูลปีของหนัง เนื่องจากใน dataset จะมีหนังบางเรื่องที่ ไม่มีข้อมูลวันที่หนังเริ่มฉายจึงต้องมีการ impute ข้อมูลโดยการแทนด้วย 0 จากนั้นจึงสามารถ filter ปีที่ฉายตั้งแต่ปี 2001 เป็นต้นไปได้

หลังจากนั้นจึงเลือกเฉพาะหนังที่มีการถ่ายทำหรือผลิตขึ้นใน US เท่านั้น



รูปที่ 3 จำนวนของหนังแบ่งตามปีที่ฉายหลัง filter ทั้งหมด

สำหรับ dataset crew.csv ที่ได้จากการดึง API มาจะเป็นข้อมูลที่มีลักษณะ record ที่แต่ละแถวมีข้อมูลหนังและนักแสดง หากนำมาสร้างเป็นกราฟก็จะเป็นกราฟที่มีลักษณะ Bipartite คือมีจุด 2 ประเภท คือ จุดหนัง และจุดนักแสดง จะต้องทำการแปลงเป็นกราฟที่มีลักษณะ Unipartite โดยกำหนดให้จุดแต่ละจุดคือ นักแสดง มีเส้นที่เชื่อมกันบอกถึงความสัมพันธ์ของจุดนักแสดงสองจุดที่เคยเล่นหนังด้วยกัน (Co-Starring) ในส่วนของวิธีการ transform data จาก actor-movie ไปเป็น actor-actor นั้นโดยการสร้าง object ที่เป็นตาราง 2 column คือ column movie กับ column list ที่รวม actor ที่เล่นหนังเรื่องนั้นทุกคน จากนั้นทำการจับคู่หา combination ของ actor 2 คนในหนังเรื่องนั้นทั้งหมด และเรียงชื่อ actor ตามลำดับตัวอักษรเพื่อจะสามารถ aggregate count คู่ของนักแสดงคู่ที่เคยเล่นหนังด้วยกันมากกว่า 1 ครั้ง โดยจะใช้ข้อมูลนี้สร้างเป็น column น้ำหนักในการสร้างกราฟเครือข่าย หลังจากนั้นเอาผลลัพธ์ที่ได้จากการทำการจับคู่ไปทำเป็น table edge เพื่อนำไปสร้างเป็นกราฟเครือข่าย ซึ่งข้อมูลที่ได้จากการ cleaning data ทั้งหมดแล้วจะประกอบไปด้วยข้อมูลของนักแสดงทั้งหมด 864 คน จากหนังทั้งหมด 1,091 เรื่องที่อยู่ในช่วงเวลาปีค.ศ. 2001 เป็นต้นไป ซึ่งเมื่อนำมาสร้างเป็น network graph เราจะได้จุดของนักแสดงทั้งหมด 864 จุดและจำนวนเส้นเชื่อมของนักแสดงที่เล่นหนังด้วยกันทั้งหมด 10,800 เส้นเชื่อม

4. การวิเคราะห์ข้อมูล

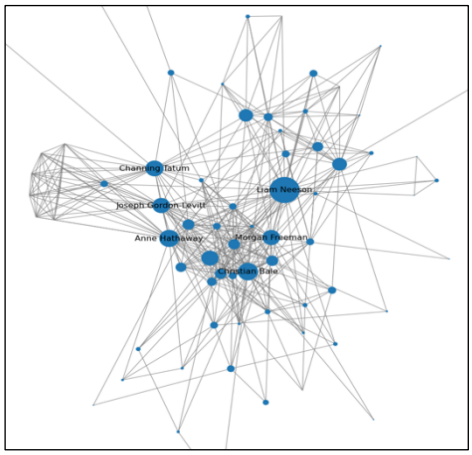
ในการวิเคราะห์ข้อมูลเราเริ่มจากการหาความสัมพันธ์ของ network graph โดยดูได้จากค่า Degree Centrality, Closeness Centrality, Betweenness Centrality, Eigenvector Centrality รวมถึงการหาค่า Clustering Coefficient เพื่อให้ทราบถึงความเป็นศูนย์กลางของเครือข่ายซึ่งในงานวิจัยนี้เราจะเน้นการวิเคราะห์ข้อมูลในลักษณะ Betweenness Centrality เพื่อให้ทราบว่านักแสดงคนไหนที่มีค่า Betweenness มากจะแสดงว่านักแสดงคนนั้นเป็นตัวกลางในการเชื่อมต่อของนักแสดงกลุ่มหนึ่งไปอีกกลุ่มหนึ่งได้ดี

Rank	Sort by Degree Centrality		Sort by Clustering Coefficient		Sort by Eigenvector Centrality		Sort by Closeness Centrality		Sort by Betweenness Centrality	
	Name	Deg_Central	Name	Clust_Coeff	Name	Ev_Cent	Name	Close_Cent	Name	Btw_Cent
1	Stan Lee	0.1611	Stan Lee	0.5158	Stan Lee	0.1678	Stan Lee	0.5158	Samuel L. Jackson	0.0317
2	Samuel L. Jackson	0.1541	Liam Neeson	0.5059	Samuel L. Jackson	0.1574	Liam Neeson	0.5059	Hugh Jackman	0.0265
3	Liam Neeson	0.1483	Samuel L. Jackson	0.5053	Josh Brolin	0.1463	Samuel L. Jackson	0.5053	Natalie Portman	0.0248
4	Scarlett Johansson	0.1333	Josh Brolin	0.5015	Scarlett Johansson	0.1428	Josh Brolin	0.5015	Liam Neeson	0.0248
5	Josh Brolin	0.1263	Scarlett Johansson	0.4983	Tilda Swinton	0.1338	Scarlett Johansson	0.4983	Dwayne Johnson	0.0206
6	Matt Damon	0.1205	Tilda Swinton	0.4929	Chris Hemsworth	0.1307	Tilda Swinton	0.4929	Stan Lee	0.0189
7	Tilda Swinton	0.1170	Matt Damon	0.4906	Chris Evans	0.1250	Matt Damon	0.4906	Ryan Reynolds	0.0182
8	Chris Hemsworth	0.1078	Jude Law	0.4881	Robert Downey Jr.	0.1239	Jude Law	0.4881	Jason Statham	0.0169
9	Jon Favreau	0.1078	Paul Rudd	0.4846	Benedict Cumberbı	0.1238	Paul Rudd	0.4846	Tilda Swinton	0.0165
10	Chris Evans	0.1066	Ryan Reynolds	0.4846	Jeremy Renner	0.1220	Ryan Reynolds	0.4846	Scarlett Johansson	0.0157
11	Jude Law	0.1054	Chris Pratt	0.4829	Natalie Portman	0.1217	Chris Pratt	0.4829	Vince Vaughn	0.0152
12	Robert Downey Jr.	0.1020	Jon Favreau	0.4829	Jon Favreau	0.1216	Jon Favreau	0.4829	Bruce Willis	0.0143
13	Natalie Portman	0.1020	Bruce Willis	0.4819	Chris Pratt	0.1206	Bruce Willis	0.4819	Matt Damon	0.0139
14	Ryan Reynolds	0.1020	Mark Strong	0.4816	Zoe Saldana	0.1198	Mark Strong	0.4816	Ben Affleck	0.0138
15	Anne Hathaway	0.1008	Chris Hemsworth	0.4813	Paul Rudd	0.1185	Chris Hemsworth	0.4813	Channing Tatum	0.0132
16	Chris Pratt	0.0985	Anne Hathaway	0.4810	William Hurt	0.1184	Anne Hathaway	0.4810	Ben Stiller	0.0130
17	Ben Affleck	0.0985	Hugh Jackman	0.4802	Gwyneth Paltrow	0.1156	Hugh Jackman	0.4802	Jason Bateman	0.0126
18	Paul Rudd	0.0985	Chris Evans	0.4802	Vin Diesel	0.1107	Chris Evans	0.4802	Justin Theroux	0.0120
19	Hugh Jackman	0.0973	Ben Affleck	0.4794	Sebastian Stan	0.1091	Ben Affleck	0.4794	Josh Brolin	0.0117
20	Channing Tatum	0.0950	Christian Bale	0.4794	Marisa Tomei	0.1090	Christian Bale	0.4794	Mark Wahlberg	0.0115

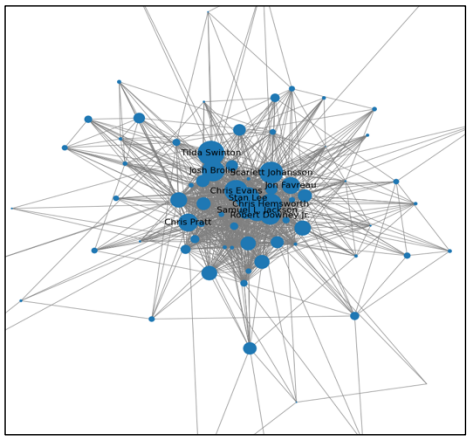
ตารางที่ 1 แสดงผลการคำนวณค่า Centrality

จากตารางที่ 1 จะเห็นได้ว่าการคำนวณค่า Centrality ของแต่ละตัวมีค่าและความหมายแตกต่างกันออกไป เนื่องจากค่าที่นำมาใช้ในการคำนวณ centrality ของแต่ละเครื่องมือวัดนั้นแตกต่างกัน อาทิเช่น Degree Centrality ซึ่งคำนวณจากสัดส่วน degree ของ node i เทียบกับจำนวน node ทั้งหมด, Clustering Coefficient ซึ่งคำนวณจากอัตราส่วนระหว่างจำนวน edge เฉพาะเพื่อนบ้านของ node i เทียบกับ degree ของ node i, Closeness Centrality ซึ่งคำนวณจากผลรวมระยะทางของเส้นทางที่สั้นที่สุดระหว่าง node i ไปยัง node ที่เหลือทั้งหมด เป็นต้น ซึ่งค่า centrality เหล่านั้นอาจจะไม่เหมาะกับบริบทของการวิเคราะห์เครือข่ายของรายงานนี้ เนื่องจากการใช้จำนวน degree ค่าผลรวมระยะทางเส้นทางที่สั้นที่สุด หรือการใช้จำนวน edge เฉพาะระหว่างเพื่อนบ้านในการหาว่านักแสดงคนไหนเป็นจุดศูนย์กลางของการแสดงหนังร่วมทั้งหมดอาจไม่เหมาะสมพอ ดังนั้นในรายงานนี้เราให้ความสนใจกับค่า Betweenness Centrality เนื่องจากเป็นการคำนวณเพื่อหาว่านักแสดงคนไหนเป็น node ทางผ่านในการเชื่อมกันของนักแสดงกลุ่มอื่น ๆ ได้มาก ยกตัวอย่างเช่น Samuel L. Jackson ที่เป็นนักแสดงที่ผ่านการเล่นหนังมาหลายเรื่อง และแสดงหนังร่วมกับกับกลุ่มของนักแสดงหลากหลายกลุ่ม ทำให้เขามีค่า Betweenness Centrality ที่มากที่สุด

ในขั้นตอนถัดไปจะเป็นการทำ Community Detection เพื่อหาว่านักแสดงคนไหนบ้างที่อยู่ใน community เดียวกัน มีการจับกลุ่มกันหนาแน่น แสดงว่านักแสดงคนนั้นเคยร่วมงานกันบ่อยครั้ง น่าจะทำงานร่วมกันได้ดีในหนังเรื่องถัดไป การทำ Detection Community ในรายงานนี้จะใช้ Louvain Algorithm ซึ่งใช้การคำนวณค่า modularity โดยผลลัพธ์สามารถแบ่งกลุ่มออกมาได้ทั้งหมด 11 กลุ่ม ซึ่งแสดงเป็น network graph ที่มี node แต่ละ node แทนนักแสดงแต่ละคน มีขนาดของ node ใหญ่ตามค่า Betweenness Centrality ซึ่งแต่ละกลุ่มจะมีการเชื่อมต่อที่หนาแน่นแตกต่างกันออกไป ยกตัวอย่างเช่นนักแสดงใน community กลุ่มที่ 1 ที่เป็นกลุ่มนักแสดงนำของหนังตลาดทั่วไป กับ community กลุ่มที่ 7 ซึ่งเป็นกลุ่มของนักแสดงที่อยู่ในจักรวาล Marvel Cinematic Universe



รูปที่ 4 แสดงความสัมพันธ์ของนักแสดงกลุ่มที่ 1



รูปที่ 5 แสดงความสัมพันธ์ของนักแสดงกลุ่มที่ 7

จาก network graph ในรูปที่ 4 และ 5 จะเห็นว่ากลุ่มที่ 1 มีเครือข่ายที่หนาแน่นน้อยกว่ากลุ่มที่ 7 อย่างชัดเจนเนื่องจากกลุ่มที่ 7 เป็นกลุ่มของนักแสดงที่แสดงร่วมกันบ่อยครั้ง ทำให้เครือข่ายมีความหนาแน่น ซึ่งขนาดของจุดจะขึ้นอยู่กับค่า Betweenness Centrality

โดยจากการคำนวณค่าเฉลี่ยของนักแสดง 20 อันดับแรกของตัวแปรในแต่ละกลุ่ม ดังตารางที่ 2 จะเห็นได้ว่า แต่กลุ่มมีค่าตัวแปรที่มีลักษณะแตกต่างกัน

Community Group	Mean			
	Amount Movie Played	Popularity	degree	btw_cent
0	12	21	52	0.0062
1	6	15	23	0.0031
2	10	19	39	0.0048
3	7	19	34	0.0043
4	10	17	41	0.0061
5	8	23	27	0.0022
6	17	20	85	0.0095
7	10	17	42	0.0018
8	10	23	34	0.0029
9	8	24	27	0.0055
10	8	11	23	0.0010

ตารางที่ 2 แสดงผลการคำนวณค่า Betweenness แต่ละกลุ่ม

5. สรุปผล

การใช้หลักการของ Betweenness Centrality ในการหานักแสดงที่เป็นจุดศูนย์กลางในวงการหนังโดยอาศัยความสัมพันธ์จากการที่เคยแสดงร่วมกัน (Co-Starring) ดูจะเป็นวิธีที่เหมาะสมที่สุด ยิ่งมีค่า Betweenness Centrality สูงจะบ่งบอกว่าสมาชิกนั้นเป็นจุดทางผ่านในการที่จะเชื่อมนักแสดงคนอื่นเข้าด้วยกัน หรือเป็น node ที่อยู่ในเส้นทางระยะทางที่สั้นที่สุดระหว่างคู่สมาชิกอื่นเป็นจำนวนหลายครั้งเป็นการบ่งบอกว่านักแสดงคนนั้นมีความสำคัญมาก

การใช้ Louvain algorithm ในการ Detect Community ทำให้เห็นว่านักแสดงคนไหนที่เคยร่วมเล่นหนังกันบ่อย ๆ ก็จะถูกจัดอยู่ในกลุ่มเดียวกัน และแต่ละกลุ่มนั้นเมื่อสร้างเป็น Network graph จะเห็นได้ชัดเจนว่าความสัมพันธ์ของแต่ละกลุ่มว่ามีความหนาแน่นที่แตกต่างกันและมี key influencer ของแต่ละกลุ่มแตกต่างกันด้วย ซึ่งจะเป็นประโยชน์สำหรับคนภายนอกวงการหนังที่รู้จักนักแสดงน้อยได้ทราบว่านักแสดงคนใดที่เป็นนักแสดงดังหรือมีอิทธิพลในวงการหนัง เพื่อใช้ในการคัดเลือกนักแสดงสำหรับการทำ marketing ได้ โดยดูจาก Betweenness Centrality ที่ได้จากการคำนวณ

เอกสารอ้างอิง

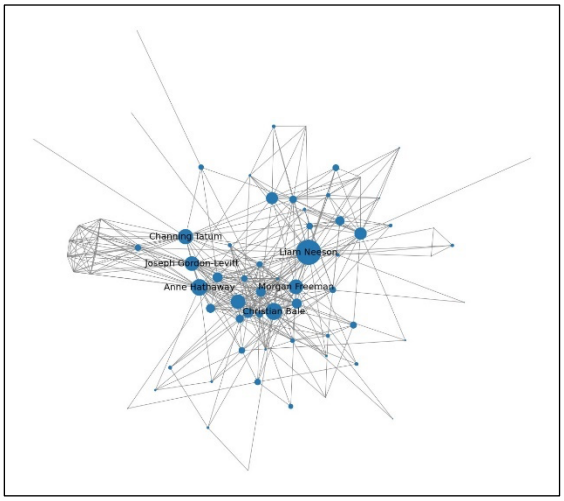
Mark E. J. Newman: Networks: An Introduction. Oxford University Press, USA, 2010, pp. 169.

Wasserman, S.; & Faust, K. (1994). Social Network Analysis: Methods and Applications. Cambridge: Cambridge University Press , pp. 177-188.

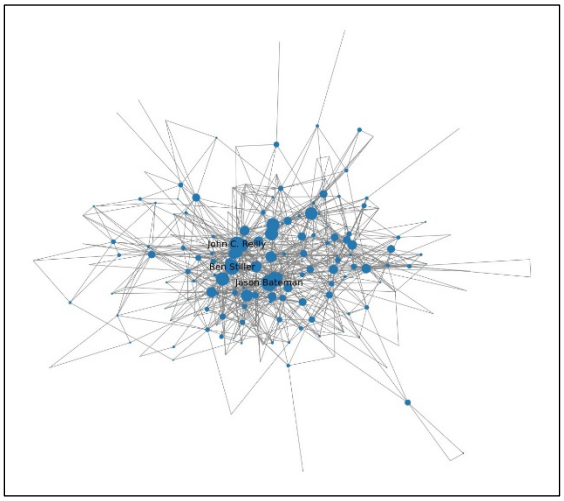
Yan, E.; & Ding, Y. (2009, October). Applying Centrality Measures to Impact Analysis: A Coauthorship Network Analysis. Journal of the American Society for Information Science and Technology.

ภาคผนวก

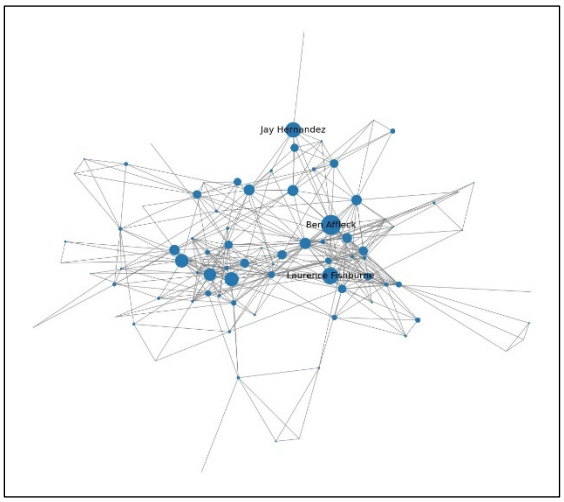
1. Community Cluster



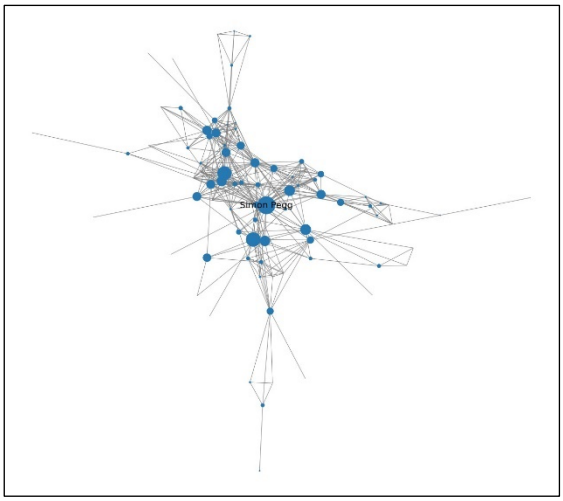
กลุ่มที่ 1 จำนวน 71 คน



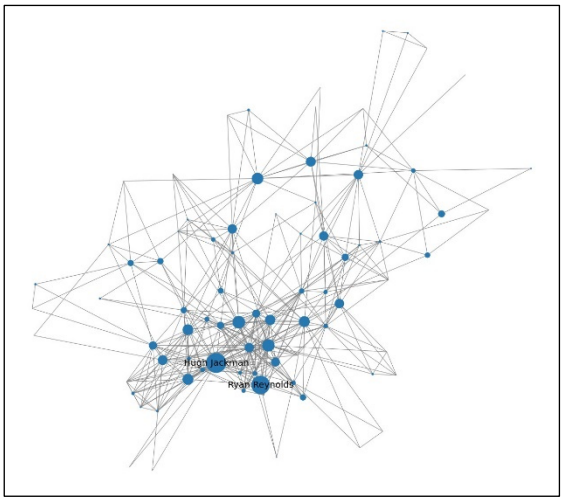
กลุ่มที่ 2 จำนวน 174 คน



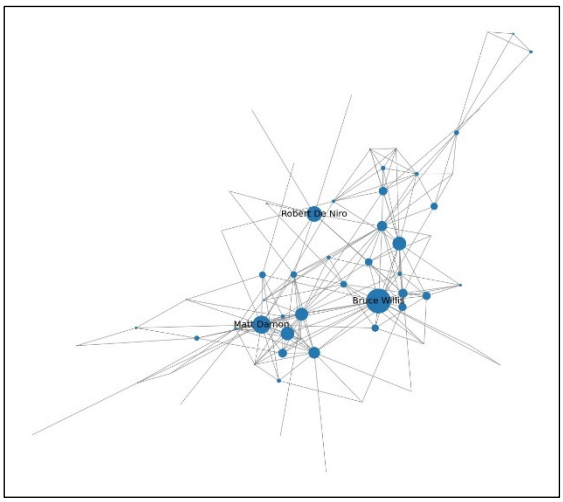
กลุ่มที่ 3 จำนวน 87 คน



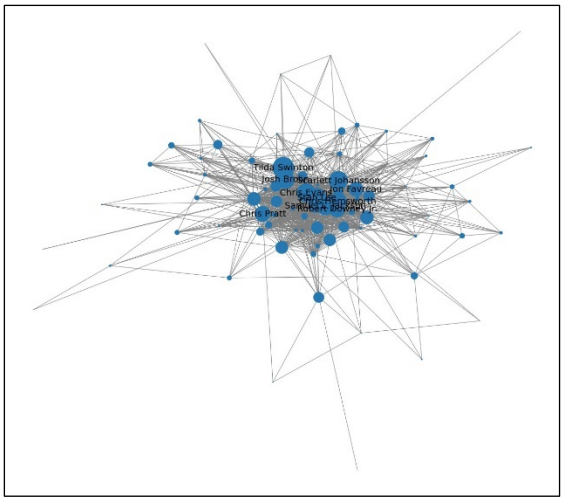
กลุ่มที่ 4 จำนวน 84 คน



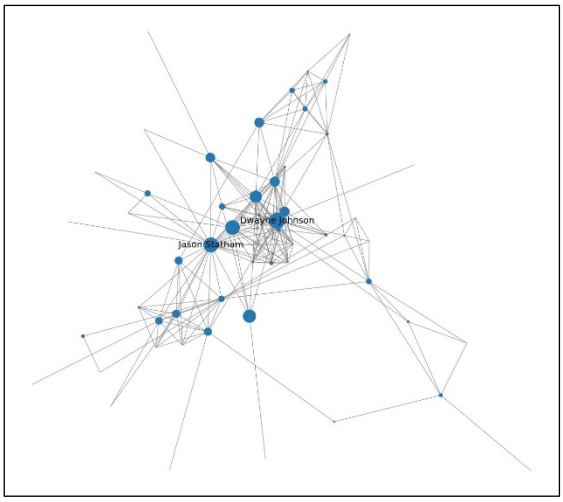
กลุ่มที่ 5 จำนวน 82 คน



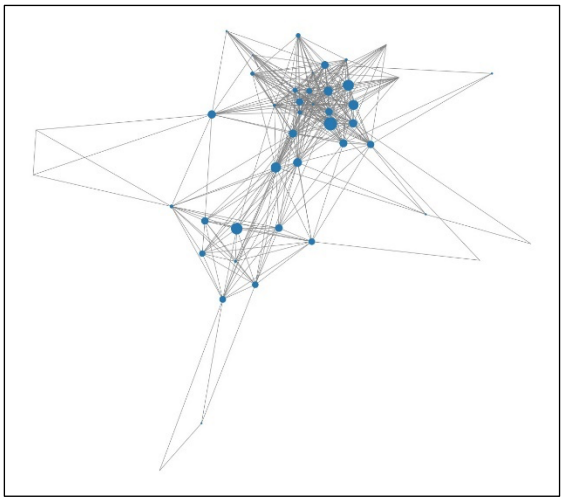
กลุ่มที่ 6 จำนวน 61 คน



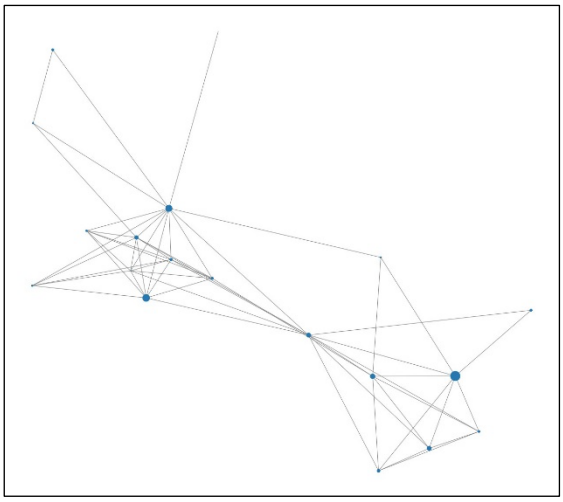
กลุ่มที่ 7 จำนวน 80 คน



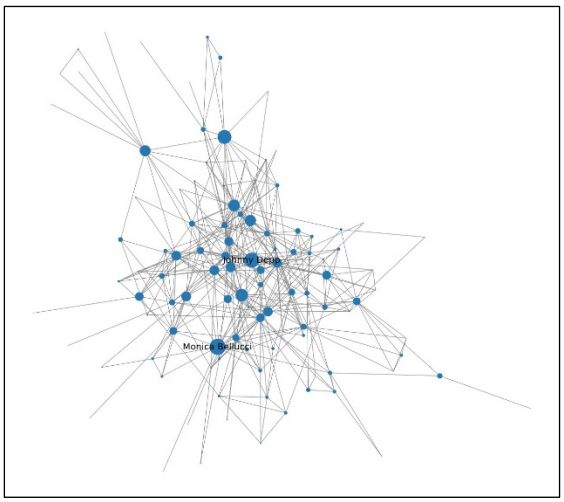
กลุ่มที่ 10 จำนวน 55 คน



กลุ่มที่ 8 จำนวน 44 คน



กลุ่มที่ 11 จำนวน 19 คน



กลุ่มที่ 9 จำนวน 107 คน

2. Python Code

dataset_preparation.ipynb

```
import pandas as pd
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
import seaborn as sns

#Create crew_actor.csv consists only Actor from crew.csv

#read crew.csv
df_crew = pd.read_csv('crew.csv')
print(df_crew.shape)
df_crew.head()

len(df_crew.name.unique())

#Read movie.csv
df_movie = pd.read_csv('movie.csv')
print(df_movie.shape)
df_movie.head()

#Filter movie to reduce crew_actor.csv size

df_movie_f1 = df_movie.loc[(df_movie['imdb_votecount'] > 10000) &
(df_movie['product_country'].str.contains('US')) & (df_movie['startYear'] >= 2001)]

#reset index
df_movie_f1.reset_index(drop=True,inplace=True)
print(df_movie_f1.shape)
df_movie_f1.head()

#create mov_list from df_movie_f1 column ['tmdb_id']
mov_list = df_movie_f1['tmdb_id'].tolist()

#Filter crew data that is in mov_list only
df_crew_f1 = df_crew.loc[df_crew['movie_id'].isin(mov_list)].reset_index(drop=True)

print(df_crew_f1.shape)
df_crew_f1.head()

#add popularity info
df_popularity = pd.read_csv('popularity_crew.csv')
print(df_popularity.shape)
df_popularity.head()

df_crew_f1 =
pd.merge(df_crew_f1,df_popularity[['id','name','popularity','gender']],on='name')
df_crew_f1

#add column edge_count of each ['name']
test =
df_crew_f1.groupby('name').agg({'name':'size'}).rename(columns={'name':'edge_count'})
df_crew_f1 = pd.merge(df_crew_f1, test, on='name', how = 'left')
df_crew_f1

#display unique job in df
df_crew_f1.job.unique()

#filter crew with specific jobs
job_list = ['Actor']
df_crew_f2 = df_crew_f1.loc[(df_crew_f1['job'].isin(job_list))]

#reset index
df_crew_f2.reset_index(drop=True,inplace=True)

df_crew_f2.head()

#Check unique name
print(len(df_crew_f2.name.unique()))
df_crew_f2.head()

#create ['actor'] column that encode ['name'] into index starting from 0
df_crew_f2['actor'] = df_crew_f2.name.astype('category').cat.codes

#Create ['movie'] column that encode ['movie_id'] into index starting from 0
df_crew_f2['movie'] = df_crew_f2.movie_id.astype('category').cat.codes

df_crew_f2

df_crew_f2.to_csv('crew_actor.csv',index=False)
```

actor_network.ipynb

```
import pandas as pd
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
import seaborn as sns
from itertools import combinations
import community
from IPython.display import display

#filter popularity >= 7, mov with > 100,000 votes, movie release date between 2001-2020

#Read movie.csv
df_movie = pd.read_csv('movie.csv')
print(df_movie.shape)
df_movie.head()

#explore movie.csv dataset of the imdb_votecount ---- too many amount of movies
sns.histplot(data=df_movie, x="imdb_votecount", bins=range(0, 1000000, 100000))

#Filter movie
#df_movie with 3 conditions: (>10000 [imdb_votecount]) & (production_country contains 'US') & (year >= year)

year = 2001     #released year of the movie to filter

df_movie_f1 = df_movie.loc[(df_movie['imdb_votecount'] > 100000) &
                           (df_movie['product_country'].str.contains('US')) &
                           (df_movie['startYear'] >= year)].reset_index(drop=True)

print(df_movie_f1.shape)
df_movie_f1.sort_values('imdb_votecount',ascending=False).head()

#check movie data distribution by counting ['imdb_cotecount']
sns.histplot(data=df_movie_f1, x="imdb_votecount", bins=range(0, 1000000, 100000))

#create mov_list from df_movie_f1 column ['tmdb_id']
mov_list = df_movie_f1['tmdb_id'].tolist()

#import crew table
df = pd.read_csv('crew_actor.csv')
print(df.shape)
df.head()

#filter popularity
df = df.loc[df['popularity'] >= 7.0]

#filter only those in mov_list
df = df.loc[df['movie_id'].isin(mov_list)]
df.shape

print('actor amount: ',len(df.actor.unique()))
print('movie amount: ',len(df.movie.unique()))

df_mov_pop = df[['name','popularity']]
df_mov_pop = df_mov_pop.drop_duplicates(subset=['name'], keep='first')
df_mov_pop.sort_values('popularity',ascending=False).head(10)

df_mov_num = df[['movie_id','name']].groupby(['name']).agg(['count'])
df_mov_num.columns = ['count']
df_mov_num.sort_values('count',ascending=False)

df['actor_1'] = df['actor'].astype(str)
df.head()

#create [movie-actor] object
mov_actor = df.groupby(['movie'])['actor_1'].apply(','.join)
mov_actor

#find combination between 2 pair of actors from [mov-actor] object
mylist = []
for i in mov_actor:
    actor_list = i.split(',')
    comb = combinations(actor_list, 2)
    for i in list(comb):
        #convert combination inside into integer so it can be sorted
        edge_pair = list(map(int, i))
        mylist.append(sorted(edge_pair))

#create edge dataframe

actor1 = []
actor2 = []
for i in mylist:
    actor1.append(i[0])
    actor2.append(i[1])

df2 = pd.DataFrame()
df2['actor1'] = actor1
df2['actor2'] = actor2
print(df2.shape)
df2.head()

#check for duplicate in edge dataframe --- the duplicated edge of the row will be aggregate into weight of the edge
print(df2[df2.duplicated(['actor1','actor2'],keep='first')].shape)

#create actor table
temp = df.sort_values(['actor'])
temp = temp.drop_duplicates(subset=['actor'], keep='first')
actor_table = temp[['actor','name']].reset_index(drop=True).copy()
print('no. of movie: ')
print('no. of actor: ', actor_table.shape)
actor_table.head()

actor_table.loc[actor_table.name == 'Orlando Bloom']

df_merged = pd.merge(df2,actor_table,left_on='actor1',right_on='actor',how='left')
df_merged.head()

df_merged = df_merged.drop(['actor'], axis=1)
df_merged.columns = ['actor1', 'actor2', 'name1']
df_merged.head()

df_merged2 =
pd.merge(df_merged,actor_table,left_on='actor2',right_on='actor',how='left')
df_merged2.head()

df_merged2 = df_merged2.drop(['actor'], axis=1)
df_merged2.columns = ['actor1', 'actor2', 'name1', 'name2']
df_merged2.head()

df_merged2['concat'] = list(zip(df_merged2.name1, df_merged2.name2))
print(df_merged2.shape)
df_merged2.head()

#create ['repeat'] col that will be used as weight of edges
temp =
df_merged2.groupby('concat').agg({'concat':'size'}).rename(columns={'name':'repeat'})
temp.columns = ['repeat']

print(temp.shape)
temp.sort_values('repeat', ascending=False)

print('pre-undupe :',df_merged2.shape)
df_merged2_undupe = df_merged2.drop_duplicates(subset=['concat'], keep='first')
print('post-undupe :',df_merged2_undupe.shape)

df_edge = pd.merge(df_merged2_undupe,temp,on='concat',how='left')
df_edge['ebunch'] = list(map(list, zip(df_edge.name1, df_edge.name2, df_edge.repeat)))
```

```
df_edge.sort_values('repeat',ascending=False)

print('no. of movie: ', len(df.movie_id.unique()))
print('no. of actor: ', len(df.name.unique()))
print('no. of actor: ', actor_table.shape)

#create function to pass the edge pair and weight into nx.add_edges_from
def edge_generator(reader):
    reader
    for row in reader:
        row[2] = float(row[2]) # convert numeric weight to float
        yield (row[0],
               row[1],
               dict(weight=row[2]))

#create undirected graph
G = nx.Graph()

#add nodes
G.add_nodes_from(actor_table.name.tolist())

#add edge pairs with weight from edge_generator function
G.add_edges_from(edge_generator(df_edge.ebunch.tolist()))

# get edge data between 2 nodes
print(G.get_edge_data('Johnny Depp', 'Keira Knightley'))

#check if graph is connected
print('Is G graph connected: ',nx.is_connected(G))
print(nx.info(G))

#remove non-connected node from G
print('list of non-connected node: ', list(nx.isolates(G)))
G.remove_nodes_from(list(nx.isolates(G)))

print('Is G graph connected: ',nx.is_connected(G))
print(nx.info(G))

#check for components within graph network -- found 2 components: main components
and 1 small component of 2 memebers
print('connect components: ', nx.connected_components(G))
for i in nx.connected_components(G):
    print(i)

#remove small component
G.remove_nodes_from(['Cynthia Nixon', 'Annaleigh Ashford'])
print('Is G graph connected: ',nx.is_connected(G))
print(nx.info(G))

#find community using Louvain Algorithm
partition = community.best_partition(G, random_state=0)

#community detection
dict_comm = dict(partition)
# values = [partition.get(node) for node in G.nodes()]

name = list(nx.nodes(G))

#create actor single view

agg = pd.DataFrame()
agg['name'] = name
agg = pd.merge(agg,df_mov_num,on='name',how='left')
agg.columns = ['name','num_mov']
agg = pd.merge(agg,df_mov_pop,on='name',how='left')

#add community_group from Louvain Algorithm
agg['comm_group'] = dict_comm.values()
num_comm = len(agg.comm_group.unique())

#check numbers of community from
len(agg.comm_group.unique())

#find degree
d = nx.degree(G)
dict_degree = dict(d)
#compute degree centrality
dc = nx.degree_centrality(G)
dict_dc = dict(dc)
#compute clustering coefficient
c = nx.clustering(G)
dict_cc = dict(c)
#compute eigen vector centrality
evc = nx.eigenvector_centrality(G,max_iter=1000)
dict_evc = dict(evc)
#compute closeness_centrality
cc = nx.closeness_centrality(G)
dict_cc = dict(cc)
#compute betweenness centrality
btw = nx.betweenness_centrality(G,k=100)
dict_btw = dict(btw)

#create actor single-view

agg = pd.DataFrame()
agg['name'] = name
agg = pd.merge(agg,df_mov_num,on='name',how='left')
agg.columns = ['name','num_mov']
agg = pd.merge(agg,df_mov_pop,on='name',how='left')
agg['comm_group'] = dict_comm.values()

agg['degree'] = dict_degree.values()
agg['deg_central'] = dict_dc.values()
agg['clust_coeff'] = dict_cc.values()
agg['ev_cent'] = dict_evc.values()
agg['close_cent'] = dict_cc.values()
agg['btw_cent'] = dict_btw.values()

# agg.to_csv('agg_Final.csv',index=False)
# from google.colab import files
# files.download('agg_Final.csv')

agg = agg.sort_values('btw_cent',ascending=False).reset_index(drop=True)
print(agg.shape)
agg.head()

#Filter by each community group and sort by Global Betweeness Centrality
sort_by = 'btw_cent'
df_list = []
for i in range (0,len(agg.comm_group.unique())):
    df_test = agg.loc[agg['comm_group'] == i].sort_values(sort_by,
ascending=False)
    df_list.append(df_test)

count = 0
for i in df_list:
    print('community_group: ', count)
    print('community size: ', len(i.index))
    display(i.head(20))
    print(i.shape)
    print()
    print()
    count += 1

#visualize community group 0, 5, 6
for i in [0,5,6]:        #range(0, len(agg.comm_group.unique()))

    #create sub_dataframe of each community group
```

```
    sub_df = agg.loc[agg['comm_group'] == i]

    #create sub_graph of each community group
    sg = G.subgraph(sub_df.name.tolist())
    mydict = dict(zip(sub_df.name,sub_df.btw_cent))

    #visualize each community group
    print('community group: ', i)
    print('community size: ', len(sub_df.index))

    hubs = sub_df.loc[sub_df['popularity'] >= 20.0].name.tolist()
    labels = {}
    for node in sg.nodes():
        if node in hubs:
            labels[node] = node

    plt.figure(3,figsize=(15,10))
    nx.draw(sg,pos = nx.spring_layout(sg, seed = 100),nodelist=mydict.keys(),
node_size=[v * 100000 for v in mydict.values()],edge_color='gray')
    nx.draw_networkx_labels(sg,pos = nx.spring_layout(sg, seed =
100),labels=labels,font_size=16,font_color='k')
    plt.show()
    print()
```