

✓ Harry Potter Universe: A Social Network Analysis

Contents:

- Detailed Description of Network
- Network Properties
- Distribution Plots
- Identifying Most Important Nodes
- Network Visualizations
- Network Type


✓ 1. Detailed Description of Network

- Number of nodes
- Number of links
- Type of network

Examined dataset (textual data transformed): <https://github.com/shecodespython/harry-potter-network-analysis>

```
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np
```


```
from google.colab import drive
drive.mount('/content/drive')
```

 Mounted at /content/drive

```
HP = nx.read_edgelist("/content/drive/MyDrive/SNA/merged_network.csv",
                    delimiter=",",
                    nodetype=str,
                    data=[('weight', str)],
                    create_using=nx.Graph())
```

Nodes and Links

```
len(HP.nodes()), len(HP.edges())
```

 (254, 606)

```
nx.is_directed(HP)
```

 False

```
nx.is_weighted(HP)
```

 True

The Harry Potter real dataset has **254 nodes** and **606 links**. As we can see, the network is **undirected** but it is **weighted**.

The **nodes** here are all the **characters** that appear throughout the whole Harry Potter book series (all 7 books), and **links** are the **relationships between them**. The links are based on whether two characters appear in a window of five sentences in the text (the text version of the respective book). The **weight** values are established based on **how many times a certain pair appears** in the before-mentioned context.

✓ 2. Network Properties - Microscale Analysis

- Diameter
- Connected components
- Average degree
- Average shortest path
- Average Clustering Coefficient
- Assortativity Coefficient

Diameter

```
diameter = nx.diameter(HP)
```

```

-----
NetworkXError                                Traceback (most recent call last)
<ipython-input-7-89fe52b10dc6> in <cell line: 1>()
----> 1 diameter = nx.diameter(HP)

----- 7 frames -----
/usr/local/lib/python3.10/dist-packages/networkx/algorithms/distance_measures.py in eccentricity(G, v, sp, weight)
    318         else:
    319             msg = "Found infinite path length because the graph is not " " connected"
--> 320             raise nx.NetworkXError(msg)
    321
    322         e[n] = max(length.values())

NetworkXError: Found infinite path length because the graph is not connected

```

We encountered a 'NetworkXError' when trying to compute the diameter of the graph, suggesting that the **graph is not connected**, meaning there are nodes in the graph that are not reachable from each other.

```
nx.is_connected(HP)
```

```
False
```

To resolve this matter, we need to find the largest connected component to work with that from this moment on.

```
num_components = nx.number_connected_components(HP)
num_components
```

```
11
```

```
component_sizes = [len(component) for component in nx.connected_components(HP)]
component_sizes
```

```
[2, 232, 2, 2, 2, 2, 3, 2, 2, 2, 3]
```

There are 11 connected components, most of them only connecting to 2-3 nodes. This means, that the largest connected component will be the one with 232 as component size.

```
average_degree = sum(dict(HP.degree()).values()) / len(HP)
average_degree
```

```
4.771653543307087
```

An average degree of 4.77 reinforces that most nodes have around 2 to 3 connections to other nodes in the network.

Now, we can extract the **largest connected component** and focus our analysis on this particular component and analyze its properties separately.

```
largest_component = max(nx.connected_components(HP), key=len)
largest_subgraph = HP.subgraph(largest_component)
```

```
len(largest_component)
```

```
232
```

As we suspected, the one with size 232 is the largest connected component.

```
HPC = largest_subgraph
```

```
len(HPC.nodes()), len(HPC.edges())
```

```
(232, 594)
```

Our **HPC** graph is still large enough for the analysis, it has **232 nodes** and **594 links**.

Network Properties of graph HPC (largest subgraph)

Diameter

```
diameter = nx.diameter(HPC)
diameter
```

↗ 9

In the context of this network representing characters of the Harry Potter Universe connected by character relationships, the diameter of 9 would mean that the longest shortest path between any two characters in terms of the number of relationships you have to traverse is 9, meaning that the longest chain of character relationships connecting any two character in the network would have 9 links.

Social networks in real life often have diameters ranging from 3 to 7. So, a diameter of 9 in our Harry Potter social network might indicate that the network is relatively large or more complex than typical **real-world social networks**.

In a completely **random network**, the diameter tends to increase logarithmically with the number of nodes. So, a diameter of 9 could indicate a relatively.

Scale-free networks, like many real-world networks, often exhibit small diameters despite their large size. They typically have a few highly connected nodes that enable efficient information flow across the network. In our Harry Potter network, a diameter of 9 might be relatively high for such a network.

Small-world networks have relatively short path lengths between nodes compared to regular random networks. A diameter of 9 might be considered moderate, indicating some degree of small-world behavior.

Average Degree

```
avg_degree = sum(dict(HPC.degree()).values()) / len(HPC)
avg_degree
```

↗ 5.120689655172414

In our new network HPC, the average degree is around 5.12, meaning that each node has around 5 connections to other nodes.

In many **real-world social networks**, the average degree varies widely depending on factors like the size of the network and the nature of social interactions. However, in small to medium-sized social networks, an average degree of 5.12 could be considered moderate, indicating that characters have several connections, but it's not excessively high.

In a **random network**, the average degree is often quite uniform across nodes. An average degree of 5.12 in our Harry Potter social network could indicate that it's more structured or organized than a completely random network.

Scale-free networks are characterized by having a few highly connected nodes and many nodes with few connections. An average degree of 5.12 might suggest that our network does not exhibit strong scale-free characteristics.

An average degree of 5.12 could contribute to the **small-world property**, particularly if coupled with a moderate clustering coefficient and a relatively short average path length.

Average Shortest Path

```
avg_shortest_path = nx.average_shortest_path_length(HPC)
avg_shortest_path
```

↗ 3.5300417972831766

In many **real-world social networks**, the average shortest path tends to be relatively short, typically ranging from 3 to 7. An average shortest path of 3.53 in our Harry Potter social network could indicate that characters are relatively closely connected, suggesting a high level of interaction and communication within the network.

In a completely **random network**, the average shortest path tends to increase logarithmically with the number of nodes. An average shortest path of 3.53 could suggest that our network is not completely random, as random networks often have longer average shortest paths.

Scale-free networks often exhibit short average shortest paths despite their large size, due to the presence of highly connected nodes that facilitate efficient information flow. An average shortest path of 3.53 could indicate that our network has some scale-free properties.

An average shortest path of 3.53 would align with the **small-world property**, suggesting that our Harry Potter social network enables efficient communication and information dissemination between characters.

Average Clustering Coefficient

```
nx.average_clustering(HPC)
```

```
0.16868414571354406
```

In many **real-world social networks**, the clustering coefficient tends to be relatively high, often exceeding 0.2 or even 0.3. A value of 0.17 suggests that while there is some clustering in our Harry Potter social network, it may not be as tightly knit as some real-world social networks.

In a completely **random network**, the average clustering coefficient tends to be very low. Random networks lack the tendency for nodes to form clusters or communities. Therefore, an average clustering coefficient of 0.17 in our network indicates that there is indeed some degree of non-randomness in the connections between characters.

Scale-free networks often exhibit high clustering coefficients due to the presence of highly connected nodes that form clusters around them. An average clustering coefficient of 0.17 could suggest that this network does not strongly exhibit scale-free properties, as the clustering coefficient is relatively modest.

An average clustering coefficient of 0.17 aligns with the **small-world property**, indicating some degree of local clustering despite relatively short average shortest paths between characters.

Assortativity Coefficient

```
nx.degree_assortativity_coefficient(HPC)
```

```
-0.028283631710740935
```

The assortativity coefficient quantifies the tendency of nodes in a network to connect to other nodes that are similar in some way. For **real-world networks**, a negative assortativity coefficient suggests disassortative mixing, meaning nodes with different characteristics are more likely to connect to each other.

In a completely **random network**, the assortativity coefficient is typically close to zero, indicating no particular preference for nodes to connect based on any characteristic. A negative assortativity coefficient in our network suggests that there is some level of non-randomness in the connections, where nodes with different characteristics tend to connect.

Scale-free networks often exhibit assortative mixing, where nodes with similar characteristics tend to connect to each other. A negative assortativity coefficient might suggest that our network does not strongly exhibit scale-free properties, as it indicates a tendency for nodes with different characteristics to connect.

Small-world networks can have assortativity coefficients that vary depending on their specific characteristics. In this Harry Potter social network, a negative assortativity coefficient could indicate that despite the short average shortest paths, there's a tendency for characters with different characteristics to connect, contributing to the overall small-world property of the network.

In summary,

- The network has a relatively large diameter of 9, indicating that it may not be as tightly connected as some real-world social networks.
- The average degree of 5.12 suggests a moderate level of connectivity, with characters having several connections on average.
- The average shortest path value of 3.53 indicates relatively efficient communication and information flow between characters.
- The average clustering coefficient of 0.17 suggests some degree of local clustering within the network.
- The assortativity coefficient value of -0.028 indicates disassortative mixing, where characters with different characteristics are more likely to connect.

Considering these characteristics, **the network exhibits properties of a small-world network**. It has short average shortest paths, indicating efficient communication, and some degree of local clustering, reflecting the formation of communities or groups within the network. Additionally, the disassortative mixing suggests diverse interactions across different groups or characteristics, which is characteristic of small-world networks where there's a balance between local clustering and global connectivity.

3. Distributions Plotting

- Degree Distribution
- Clustering Coefficient Distribution
- Degree Centrality Distribution
- Betweenness Centrality Distribution
- Eigenvector Centrality Distribution
- Connected Component Size Distribution

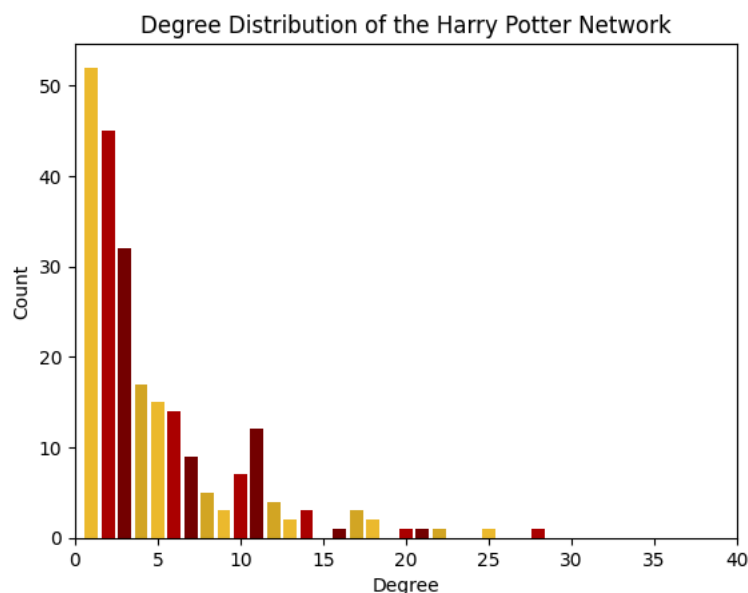
```
import collections
```

We can define some color palettes to use moving forward for our plots. These are constructed with the colors of the famous Hogwarts Houses.

```
griff_colors = ['#740001', '#ae0001', '#eeba30', '#d3a625']
slyth_colors = ['#1a472a', '#2a623d', '#5d5d5d', 'aaaaaa']
raven_colors = ['#0e1a40', '#222f5b', '#bebebe', '#946b2d']
huff_colors = ['#ecb939', '#f0c75e', '#726255', '#372e29']
```

Degree Distribution

```
degree_sequence = sorted([d for n, d in HPC.degree()], reverse=True)
print(degree_sequence)
degreeCount = collections.Counter(degree_sequence)
deg, cnt = zip(*degreeCount.items())
plt.bar(deg, cnt, width=0.80, color=griff_colors)
plt.title("Degree Distribution of the Harry Potter Network")
plt.ylabel("Count")
plt.xlabel("Degree")
plt.xlim(0, 40)
plt.show()
plt.savefig('degree_distribution.png')
```

$$\rightarrow [56, 28, 25, 22, 21, 20, 18, 18, 17, 17, 17, 16, 14, 14, 14, 13, 13, 12, 12, 12, 12, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11]$$


<Figure size 640x480 with 0 Axes>

This visualization gives insight into the distribution of node degrees within the network, which is useful for analyzing the **structure of the network**.

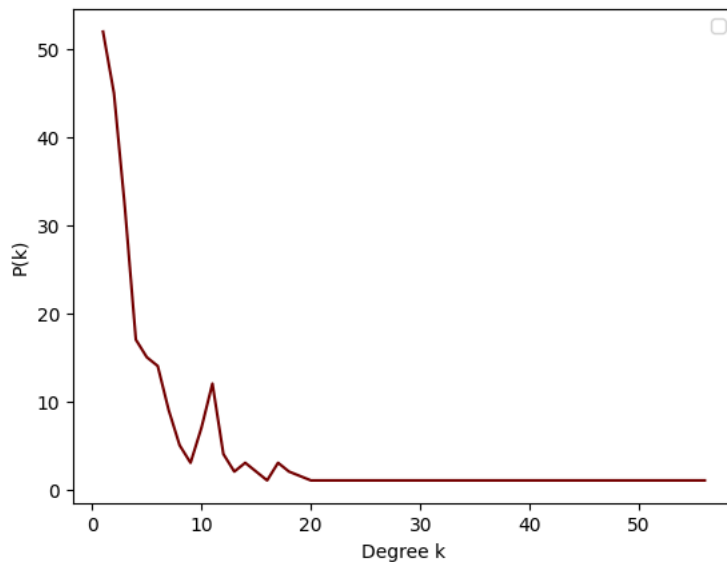
The **decreasing trend** in the histogram indicates that as the degree value increases, the number of nodes with that degree decreases. This suggests that there are fewer nodes with higher degrees, which could indicate the presence of a **few hubs** or highly connected nodes, but they

are relatively rare compared to the nodes with degree one.

Based on this degree distribution plot, the network **resembles the scale-free network structure**, which would have a long tail on the right with a few highly connected hubs.

```
plt.plot(deg, cnt, griff_colors[0])
plt.legend()
plt.xlabel('Degree k')
plt.ylabel('P(k)')
plt.show()
plt.savefig('degree_distribution2.png')
```

WARNING:matplotlib.legend.No artists with labels found to put in legend. Note that artists whose label start with an underscore are

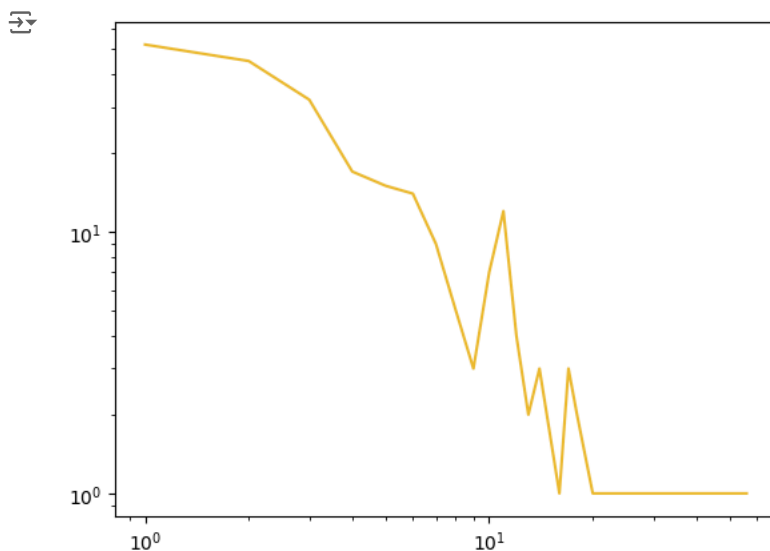


The x-axis represents the **degrees (k)** of nodes in the graph, and the y-axis represents the **probability distribution P(k)** (or the count of nodes with each degree normalized by the total number of nodes).

The **abrupt decrease** in the plot between degrees 1 and 5 indicates that there are many nodes in the network with low degrees, particularly degree 1. After the initial sharp decline, the plot becomes **almost horizontal** (after a few protuisions), indicating that the number of nodes with higher degrees remains relatively constant.

The shape of the plot suggests a **scale-free network**, common in many real-world networks like social networks and the internet.

```
plt.loglog(deg, cnt, griff_colors[2])
plt.show()
plt.savefig('degree_distribution3.png')
```

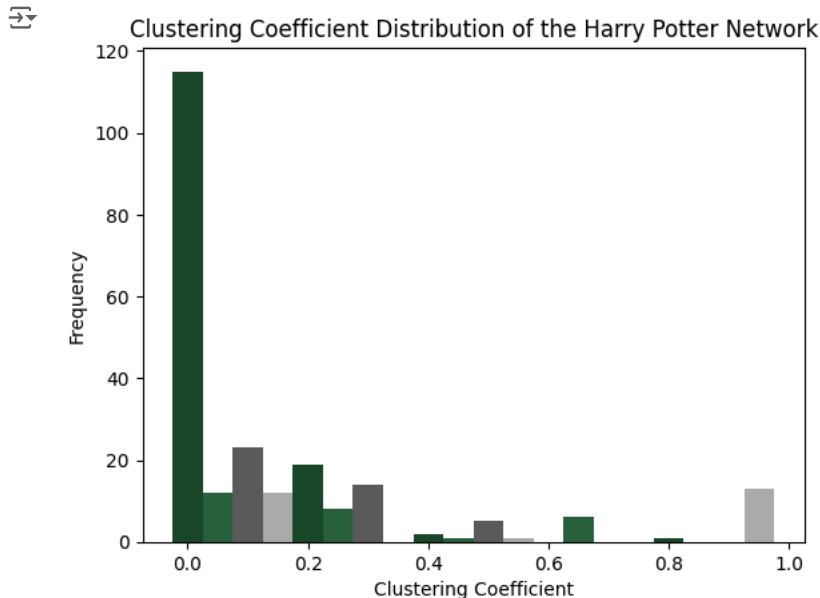


<Figure size 640x480 with 0 Axes>

We can visualize the same thing on a **logarithmic scale** as well.

Clustering Coefficient Distribution

```
clust_coeff = nx.clustering(HPC)
num_bins = 20
hist, bin_edges = np.histogram(list(clust_coeff.values()), bins=num_bins)
plt.bar(bin_edges[:-1], hist, width=np.diff(bin_edges), color=slyth_colors)
plt.xlabel('Clustering Coefficient')
plt.ylabel('Frequency')
plt.title('Clustering Coefficient Distribution of the Harry Potter Network')
plt.show()
plt.savefig('clustering_coeff_distribution.png')
```



<Figure size 640x480 with 0 Axes>

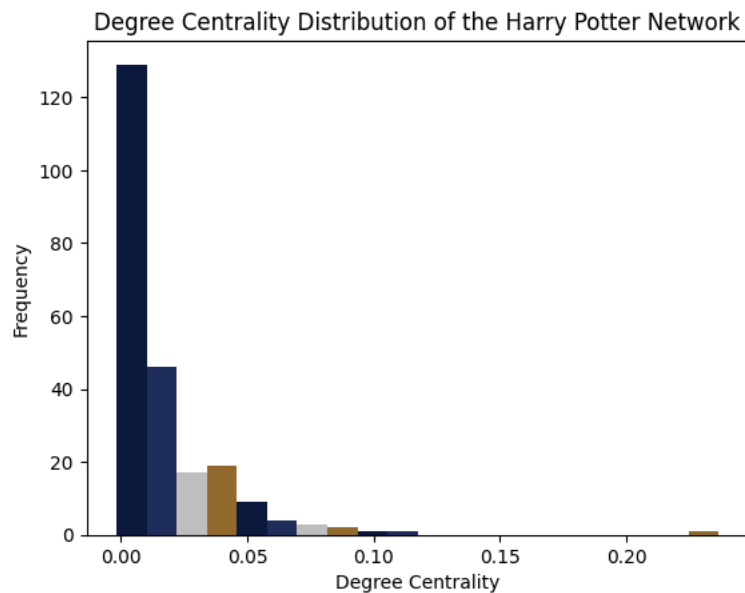
Each bin represents a **range of clustering coefficients**, and the height of each bar represents the number of nodes with a clustering coefficient falling within that range.

The very **tall first bar** indicates that there is a large number of nodes in our network with a very **low clustering coefficient**, but they are, in fact, a few nodes that tend to form clusters with other nodes.

Degree Centrality Distribution

Degree centrality is a measure of the importance of a node in a graph based on the number of edges (or connections) it has.

```
degree_centrality = nx.degree_centrality(HPC)
num_bins = 20
hist, bin_edges = np.histogram(list(degree_centrality.values()), bins=num_bins)
plt.bar(bin_edges[:-1], hist, width=np.diff(bin_edges), color=raven_colors)
plt.xlabel('Degree Centrality')
plt.ylabel('Frequency')
plt.title('Degree Centrality Distribution of the Harry Potter Network')
plt.show()
plt.savefig('degree_centrality_distribution.png')
```



<Figure size 640x480 with 0 Axes>

The x-axis represents the **degree centrality values**, which range from **0 to 1**. Nodes with a degree centrality of 0 have **no connections**, while nodes with a degree centrality of 1 are **fully connected** to all other nodes.

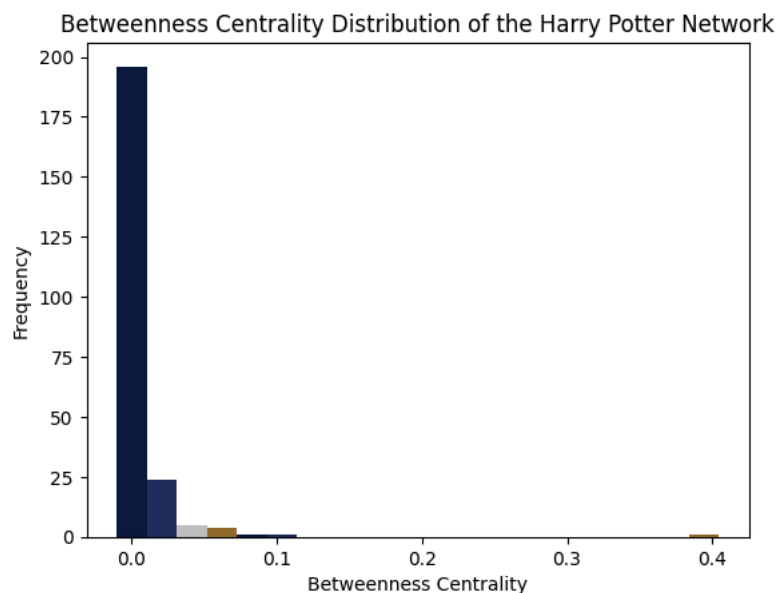
The **tall bar at the lower end** of the x-axis indicate that there is a large number of nodes in the network with **low degree centrality**. These nodes are likely located on the **periphery of the network** and have fewer connections compared to central nodes.

That node that has a degree centrality of around **0.25** is probably the main character, **Harry Potter**, who is connected to the most other characters.

Betweenness Centrality Distribution

Betweenness centrality is a measure of a node's importance based on the number of shortest paths that pass through it. Nodes with high betweenness centrality often act as bridges between different parts of the network.

```
between Centrality = nx.betweenness Centrality(HPC)
num_bins = 20
hist, bin_edges = np.histogram(list(between Centrality.values()), bins=num_bins)
plt.bar(bin_edges[:-1], hist, width=np.diff(bin_edges), color=raven_colors)
plt.xlabel('Betweenness Centrality')
plt.ylabel('Frequency')
plt.title('Betweenness Centrality Distribution of the Harry Potter Network')
plt.show()
plt.savefig('betweenness Centrality_distribution.png')
```



<Figure size 640x480 with 0 Axes>

The x-axis represents the **betweenness centrality values**, which typically range from **0 to 1**. Higher values indicate nodes that act as **important bridges** or connectors between different parts of the network.

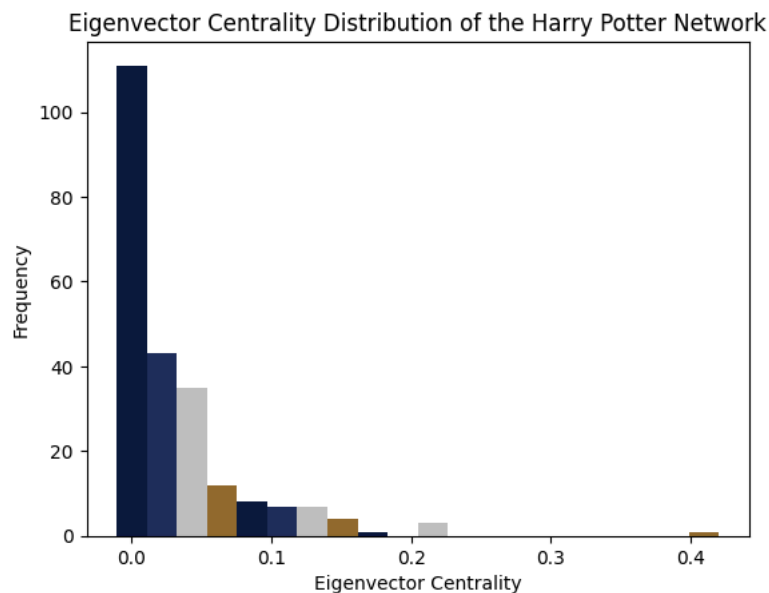
The **tall bar** at the lower end of the x-axis indicate that there is a large number of nodes in our network with **low betweenness centrality**.

Nodes with **higher betweenness centrality** are often crucial for maintaining **efficient communication** between different parts of the network.

Eigenvector Centrality Distribution

Eigenvector centrality measures the influence of a node in a network by considering both its direct connections and the importance of its neighbors.

```
eigen_centrality = nx.eigenvector centrality(HPC)
num_bins = 20
hist, bin_edges = np.histogram(list(eigen_centrality.values()), bins=num_bins)
plt.bar(bin_edges[:-1], hist, width=np.diff(bin_edges), color=raven_colors)
plt.xlabel('Eigenvector Centrality')
plt.ylabel('Frequency')
plt.title('Eigenvector Centrality Distribution of the Harry Potter Network')
plt.show()
plt.savefig('eigenvector centrality distribution.png')
```



<Figure size 640x480 with 0 Axes>

This histogram also shows a **similar** shape.

Connected Components Size Distribution

```
nx.is_connected(HPC)
```



True

```
nx.is_connected(HP)
```



False

There is no point in plotting the connected components size distribution for the HPC network since it is all connected, so we are going to conduct this part of the analysis on the **HP network**.

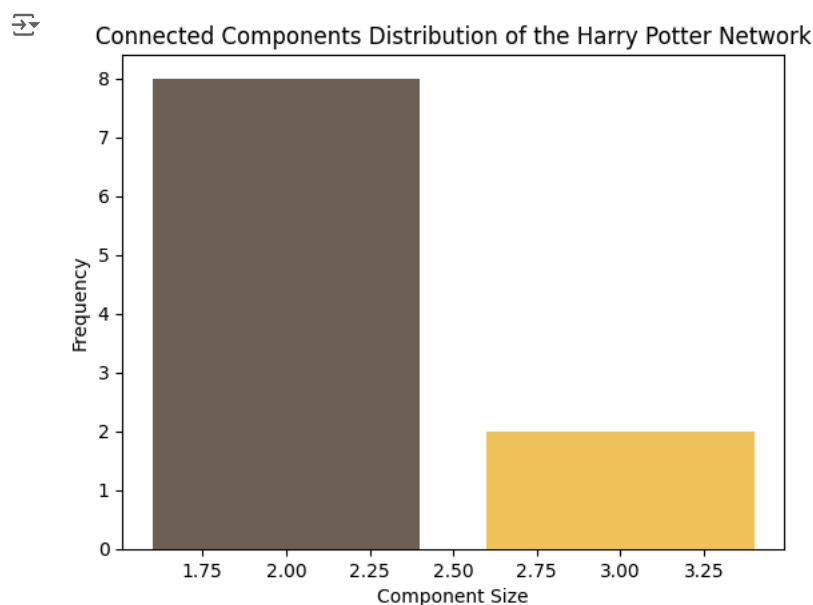
```
connected_components = nx.connected_components(HP)
component_sizes = [len(comp) for comp in connected_components]
component_sizes.sort(reverse=True)
```

```
component_count = {}
for size in component_sizes:
    if size in component_count:
        component_count[size] += 1
    else:
        component_count[size] = 1
```

```
print("Connected Components Distribution:")
for size, count in component_count.items():
    print(f"Size {size}: {count} component(s)")
```

```
↗ Connected Components Distribution:
Size 232: 1 component(s)
Size 3: 2 component(s)
Size 2: 8 component(s)
```

```
keys_to_plot = list(component_count.keys())[1:]
values_to_plot = list(component_count.values())[1:]
plt.bar(keys_to_plot, values_to_plot, color=huff_colors[1:3])
plt.xlabel('Component Size')
plt.ylabel('Frequency')
plt.title('Connected Components Distribution of the Harry Potter Network')
plt.show()
plt.savefig('connected_comp_distribution.png')
```



We **took the HPC from the plot** because it is very big compared to the rest, and nothing visible would have been displayed.

✓ 4. Identifying Most Important Nodes

To identify the most important nodes of the network, I will be comparing the top 10 nodes according to the following centrality measures:

- Degree Centrality
- Betweenness Centrality
- Closeness Centrality
- Eigenvector Centrality
- PageRank Centrality
- Harmonic Centrality
- Load Centrality

```
len(HPC.nodes()), len(HPC.edges())
```

```
↗ (232, 594)
```

Comparing different centrality measures to find the most important nodes

Degree Centrality

- Degree centrality measures the number of edges connected to a node.
- Nodes with higher degree centrality are more connected within the network, indicating a higher level of influence or importance. In a social network, nodes with high degree centrality may represent individuals with many connections or friends.
- In our particular case, if a node has a high degree centrality, it means that the given character has some kind of a relationship with many others, making him/her more important in the storyline.

```
degree_cent= nx.degree_centrality(HPC)
degree_cent
```

```
{'Adalbert Waffling': 0.012987012987012988,
 'Bathilda Bagshot': 0.047619047619047616,
 'Miranda Goshawk': 0.012987012987012988,
 'Newt Scamander': 0.008658008658008658,
 'Adrian Pucey': 0.017316017316017316,
 'George Weasley': 0.06060606060606061,
 'Lee Jordan': 0.09090909090909091,
 'Marcus Flint': 0.030303030303030304,
 'Terence Higgs': 0.008658008658008658,
 'Agatha Timms': 0.008658008658008658,
 'Ludo Bagman': 0.05627705627705628,
 'Roddy Pontner': 0.008658008658008658,
 'Aidan Lynch': 0.008658008658008658,
 'Connolly': 0.008658008658008658,
 'Quigley': 0.012987012987012988,
 'Albus Dumbledore': 0.08658008658008658,
 'Augusta Longbottom': 0.004329004329004329,
 'Avery': 0.025974025974025976,
 'Billy Stubbs': 0.004329004329004329,
 'Charity Burbage': 0.008658008658008658,
 'Dobby': 0.09523809523809523,
 'Draco Malfoy': 0.10822510822510822,
 'Gellert Grindelwald': 0.008658008658008658,
 'Griselda Marchbanks': 0.012987012987012988,
 'Harry Potter': 0.24242424242424243,
 'Karkus': 0.008658008658008658,
 'Mad-Eye Moody': 0.047619047619047616,
 'Nicolas Flamel': 0.021645021645021644,
 'Pansy Parkinson': 0.047619047619047616,
 'Percy Weasley': 0.05627705627705628,
 'Rita Skeeter': 0.0735930735930736,
 'Rubeus Hagrid': 0.017316017316017316,
 'Severus Snape': 0.04329004329004329,
 'Sirius Black': 0.07792207792207792,
 'Tiberius Ogden': 0.008658008658008658,
 'Xenophilius Lovegood': 0.021645021645021644,
 'Alecto Carrow': 0.008658008658008658,
 'Merlin': 0.030303030303030304,
 'Alice Longbottom': 0.008658008658008658,
 'Emmeline Vance': 0.025974025974025976,
 'Marlene McKinnon': 0.008658008658008658,
 'Alicia Spinnet': 0.047619047619047616,
 'Angelina Johnson': 0.030303030303030304,
 'Cho Chang': 0.04329004329004329,
 'Colin Creevey': 0.05194805194805195,
 'Geoffrey Hooper': 0.004329004329004329,
 'Katie Bell': 0.047619047619047616,
 'Luna Lovegood': 0.06060606060606061,
 'Miles Bletchley': 0.008658008658008658,
 'Warrington': 0.04329004329004329,
 'Ambrosius Flume': 0.008658008658008658,
 'Barnabas Cuffe': 0.008658008658008658,
 'Gwenog Jones': 0.004329004329004329,
 'Amos Diggory': 0.025974025974025976,
 'Apollyon Pringle': 0.004329004329004329,
 'Cedric Diggory': 0.03463203463203463,
 'Winky': 0.025974025974025976,
 'Andrew Kirke': 0.004329004329004329,
 ...}
```

```
for u, v, data in HPC.edges(data=True):
    data['weight'] = float(data.get('weight', 1))
```

Weighted Degree Centrality

```
len(HPC.nodes())
```

```
232
```

```
degree_cent = HPC.degree(weight='weight')
degree_cent = dict(degree_cent)
degree_cent = {node: centrality / (len(HPC.nodes()) - 1) for node, centrality in degree_cent.items()}
degree_cent
```

```
{'Adalbert Waffling': 0.0735930735930736,
'Bathilda Bagshot': 0.24242424242424243,
'Miranda Goshawk': 0.0735930735930736,
'Newt Scamander': 0.05194805194805195,
'Adrian Pucey': 0.09090909090909091,
'George Weasley': 0.36796536796536794,
'Lee Jordan': 0.44155844155844154,
'Marcus Flint': 0.1471861471861472,
'Terence Higgs': 0.025974025974025976,
'Agatha Timms': 0.03896103896103896,
'Ludo Bagman': 0.4025974025974026,
'Roddy Pontner': 0.03896103896103896,
'Aidan Lynch': 0.04329004329004329,
'Connolly': 0.047619047619047616,
'Quigley': 0.06060606060606061,
'Albus Dumbledore': 0.6060606060606061,
'Augusta Longbottom': 0.004329004329004329,
'Avery': 0.16017316017316016,
'Billy Stubbs': 0.008658008658008658,
'Charity Burbage': 0.017316017316017316,
'Dobby': 6.246753246753247,
'Draco Malfoy': 0.6406926406926406,
'Gellert Grindelwald': 0.0735930735930736,
'Griselda Marchbanks': 0.06060606060606061,
'Harry Potter': 6.1558441558441555,
'Karkus': 0.030303030303030304,
'Mad-Eye Moody': 0.2077922077922078,
'Nicolas Flamel': 0.05627705627705628,
'Pansy Parkinson': 0.1774891774891775,
'Percy Weasley': 0.19913419913419914,
'Rita Skeeter': 0.5238095238095238,
'Rubeus Hagrid': 0.06926406926406926,
'Severus Snape': 0.19913419913419914,
'Sirius Black': 0.4025974025974026,
'Tiberius Ogden': 0.04329004329004329,
'Xenophilius Lovegood': 0.08225108225108226,
'Alecto Carrow': 0.03463203463203463,
'Merlin': 0.16883116883116883,
'Alice Longbottom': 0.03896103896103896,
'Emmeline Vance': 0.13852813852813853,
'Marlene McKinnon': 0.03896103896103896,
'Alicia Spinnet': 0.27705627705627706,
'Angelina Johnson': 0.23376623376623376,
'Cho Chang': 0.19047619047619047,
'Colin Creevey': 0.24242424242424243,
'Geoffrey Hooper': 0.008658008658008658,
'Katie Bell': 0.41125541125541126,
'Luna Lovegood': 0.2943722943722944,
'Miles Bletchley': 0.03463203463203463,
'Warrington': 0.4025974025974026,
'Ambrosius Flume': 0.04329004329004329,
'Barnabas Cufte': 0.03463203463203463,
'Gwenog Jones': 0.021645021645021644,
'Amos Diggory': 0.15151515151515152,
'Apollyon Pringle': 0.012987012987012988,
'Cedric Diggory': 0.22077922077922077,
'Winky': 1.593073593073593,
'Andrew Kirke': 0.025974025974025976,
```

We need to find the node with the smallest degree centrality:

```
min_node = min(degree_cent, key=degree_cent.get)
min Centrality = degree_cent[min_node]
print("Node with the smallest degree centrality:", min_node)
print("Smallest degree centrality:", min Centrality)
```

```
Node with the smallest degree centrality: Augusta Longbottom
Smallest degree centrality: 0.004329004329004329
```

We need to find the node with the largest degree centrality:

```
max_node = max(degree_cent, key=degree_cent.get)
max Centrality = degree_cent[max_node]
print("Node with the largest degree centrality:", max_node)
print("Largest degree centrality:", max Centrality)
```

```
Node with the largest degree centrality: Dobby
Largest degree centrality: 6.246753246753247
```

Betweenness Centrality

- Betweenness centrality measures the extent to which a node lies on the shortest paths between other nodes in the network.
- Nodes with high betweenness centrality act as bridges or mediators in the network, facilitating communication and information flow between different parts of the network. They are important for maintaining the network's connectivity and efficiency.
- In this network, if a node has a high betweenness centrality value, it means that the given character connects others to each other, he/she is a mediator.

```
betweenness_cent = nx.betweenness_centrality(HPC)
betweenness_cent
```

```
{'Adalbert Waffling': 0.018322353974527892,
 'Bathilda Bagshot': 0.052585069535617895,
 'Miranda Goshawk': 0.010413361655597681,
 'Newt Scamander': 0.008658008658008658,
 'Adrian Pucey': 0.0036613782976899416,
 'George Weasley': 0.0222837830829147,
 'Lee Jordan': 0.0662623029906424,
 'Marcus Flint': 0.005703581269088399,
 'Terence Higgs': 0.0,
 'Agatha Timms': 0.0,
 'Ludo Bagman': 0.06168162529347615,
 'Roddy Pontner': 0.0,
 'Aidan Lynch': 0.0001672775119980089,
 'Connolly': 0.00041365281054832727,
 'Quigley': 0.008701942157672277,
 'Albus Dumbledore': 0.08147414134753661,
 'Augusta Longbottom': 0.0,
 'Avery': 0.022332854903338147,
 'Billy Stubbs': 0.0,
 'Charity Burbage': 0.00047960810945801234,
 'Dobby': 0.033699934998954954,
 'Draco Malfoy': 0.10538021482645397,
 'Gellert Grindelwald': 0.0007174025967051954,
 'Griselda Marchbanks': 0.0006033197773490725,
 'Harry Potter': 0.4148624331298386,
 'Karkus': 0.008658008658008658,
 'Mad-Eye Moody': 0.021192080278159824,
 'Nicolas Flamel': 0.012263284776633384,
 'Pansy Parkinson': 0.024803520974910252,
 'Percy Weasley': 0.04760950776945858,
 'Rita Skeeter': 0.03393794481858145,
 'Rubeus Hagrid': 0.00042367210447771635,
 'Severus Snape': 0.013361049954290001,
 'Sirius Black': 0.0876669222889753,
 'Tiberius Ogden': 0.0,
 'Xenophilius Lovegood': 0.014394056785751622,
 'Alecto Carrow': 0.000509430957898928,
 'Merlin': 0.026772860791553353,
 'Alice Longbottom': 0.007408733869855883,
 'Emmeline Vance': 0.025168633129126095,
 'Marlene McKinnon': 0.00038396386222473174,
 'Alicia Spinnet': 0.01893307897338197,
 'Angelina Johnson': 0.0012430735523481997,
 'Cho Chang': 0.007855929979163196,
 'Colin Creevey': 0.016126211679866328,
 'Geoffrey Hooper': 0.0,
 'Katie Bell': 0.00985494321273812,
 'Luna Lovegood': 0.01813996296374834,
 'Miles Bletchley': 0.00025147119020636804,
 'Warrington': 0.016449979376672516,
 'Ambrosius Flume': 0.008658008658008658,
 'Barnabas Cuffe': 0.017240730284208545,
 'Gwenog Jones': 0.0,
 'Amos Diggory': 0.01592553325965875,
 'Apollyon Pringle': 0.0,
 'Cedric Diggory': 0.008356057342713297,
 'Winky': 0.0042400081931882674,
 'Andrew Kirke': 0.0,
 ...}
```

Weighted Betweenness Centrality

```
betweenness_cent = nx.betweenness_centrality(HPC, weight='weight')
betweenness_cent
```

```
{'Adalbert Waffling': 0.01769245247506117,
 'Bathilda Bagshot': 0.046153011929409446,
 'Miranda Goshawk': 0.013141351402220967,
 'Newt Scamander': 0.008658008658008658,
 'Adrian Pucey': 0.0,
 'George Weasley': 0.03917578437623611,
 'Lee Jordan': 0.1451124733225241,
 'Marcus Flint': 0.009191291799987452,
 'Terence Higgs': 8.15609511261685e-05,
 ...}
```

```
'Agatha Timms': 0.0,
'Ludo Bagman': 0.05027605244996549,
'Roddy Pontner': 0.0,
'Aidan Lynch': 0.0004579961101700231,
'Connolly': 0.0,
'Quigley': 0.00929794842838321,
'Albus Dumbledore': 0.11094162566212251,
'Augusta Longbottom': 0.0,
'Avery': 0.019411506368028103,
'Billy Stubbs': 0.0,
'Charity Burbage': 0.016086329129807397,
'Dobby': 0.047800088110647114,
'Draco Malfoy': 0.14828137794823273,
'Gellert Grindelwald': 0.0,
'Griselda Marchbanks': 0.0,
'Harry Potter': 0.29269620836532173,
'Karkus': 0.008658008658008658,
'Mad-Eye Moody': 0.04523809523809523,
'Nicolas Flamel': 0.020715226802183333,
'Pansy Parkinson': 0.05604818984265624,
'Percy Weasley': 0.15623807242384313,
'Rita Skeeter': 0.04909815167579144,
'Rubeus Hagrid': 0.005223037831733484,
'Severus Snape': 0.06942439642044389,
'Sirius Black': 0.10477276503532854,
'Tiberius Ogden': 0.0,
'Xenophilius Lovegood': 0.017710960537047498,
'Alecto Carrow': 0.000981554677206851,
'Merlin': 0.019857582031495073,
'Alice Longbottom': 0.008563899868247694,
'Emmeline Vance': 0.025390551477508,
'Marlene McKinnon': 0.00015057406361754187,
'Alicia Spinnet': 0.01495890582847104,
'Angelina Johnson': 7.528703180877094e-05,
'Cho Chang': 0.008450393323346453,
'Colin Creevey': 0.014505704785207888,
'Geoffrey Hooper': 0.0,
'Katie Bell': 0.0015657910067847951,
'Luna Lovegood': 0.12223620221926258,
'Miles Bletchley': 0.0004266265135830352,
'Warrington': 0.01108538804190978,
'Ambrosius Flume': 0.008658008658008658,
'Barnabas Cuffe': 0.017240730284208545,
'Gwenog Jones': 0.0,
'Amos Diggory': 0.016814103770625508,
'Apollyon Pringle': 0.0,
'Cedric Diggory': 0.009843779408996797,
'Winky': 0.0004856013551665725,
'Andromeda Tonks': 0.0
```

We need to find the node with the smallest betweenness centrality:

```
min_node_betweenness = min(betweenness_cent, key=betweenness_cent.get)
min_centrality_betweenness = betweenness_cent[min_node_betweenness]
print("Node with the smallest betweenness centrality:", min_node_betweenness)
print("Smallest betweenness centrality:", min_centrality_betweenness)
```

```
→ Node with the smallest betweenness centrality: Adrian Pucey
Smallest betweenness centrality: 0.0
```

We need to find the node with the largest betweenness centrality:

```
max_node_betweenness = max(betweenness_cent, key=betweenness_cent.get)
max_centrality_betweenness = betweenness_cent[max_node_betweenness]
print("Node with the largest betweenness centrality:", max_node_betweenness)
print("Largest betweenness centrality:", max_centrality_betweenness)
```

```
→ Node with the largest betweenness centrality: Harry Potter
Largest betweenness centrality: 0.29269620836532173
```

Closeness Centrality

- Closeness centrality measures the average distance of a node to all other nodes in the network.
- Nodes with high closeness centrality are close to many other nodes in the network, indicating their ability to quickly interact or exchange information with other nodes. They are often considered to be central in terms of communication efficiency.
- In our case, if a node has a high closeness centrality value, it means that the given character can easily interact with most of the others or can easily find a mediator to reach someone else.

```
closeness_cent = nx.closeness centrality(HPC)
closeness_cent
```

```

{
  'Adalbert Waffling': 0.264,
  'Bathilda Bagshot': 0.3521341463414634,
  'Miranda Goshawk': 0.2711267605633803,
  'Newt Scamander': 0.20942883046237534,
  'Adrian Pucey': 0.2866004962779156,
  'George Weasley': 0.3678343949044586,
  'Lee Jordan': 0.38308457711442784,
  'Marcus Flint': 0.2969151670951157,
  'Terence Higgs': 0.23076923076923078,
  'Agatha Timms': 0.2679814385150812,
  'Ludo Bagman': 0.36492890995260663,
  'Roddy Pontner': 0.2679814385150812,
  'Aidan Lynch': 0.17879256965944273,
  'Connolly': 0.1834789515488483,
  'Quigley': 0.21710526315789475,
  'Albus Dumbledore': 0.39086294416243655,
  'Augusta Longbottom': 0.28136419001218027,
  'Avery': 0.30078125,
  'Billy Stubbs': 0.28136419001218027,
  'Charity Burbage': 0.30677290836653387,
  'Dobby': 0.39487179487179486,
  'Draco Malfoy': 0.41696750902527074,
  'Gellert Grindelwald': 0.29093198992443325,
  'Griselda Marchbanks': 0.3035479632063075,
  'Harry Potter': 0.49044585987261147,
  'Karkus': 0.28205128205128205,
  'Mad-Eye Moody': 0.3707865168539326,
  'Nicolas Flamel': 0.31048387096774194,
  'Pansy Parkinson': 0.3780687397708674,
  'Percy Weasley': 0.38181818181818183,
  'Rita Skeeter': 0.3888888888888889,
  'Rubeus Hagrid': 0.32038834951456313,
  'Severus Snape': 0.3737864077669903,
  'Sirius Black': 0.3921901528013582,
  'Tiberius Ogden': 0.2817073170731707,
  'Xenophilius Lovegood': 0.3447761194029851,
  'Alecto Carrow': 0.28068043742405835,
  'Merlin': 0.29277566539923955,
  'Alice Longbottom': 0.23356926188068755,
  'Emmeline Vance': 0.30196078431372547,
  'Marlene McKinnon': 0.19106699751861042,
  'Alicia Spinnet': 0.3177441540577717,
  'Angelina Johnson': 0.29806451612903223,
  'Cho Chang': 0.3276595744680851,
  'Colin Creevey': 0.37318255250403876,
  'Geoffrey Hooper': 0.2413793103448276,
  'Katie Bell': 0.3643533123028391,
  'Luna Lovegood': 0.3793103448275862,
  'Miles Bletchley': 0.2418848167539267,
  'Warrington': 0.3276595744680851,
  'Ambrosius Flume': 0.1925,
  'Barnabas Cuffe': 0.23765432098765432,
  'Gwenog Jones': 0.16153846153846155,
  'Amos Diggory': 0.34323922734026746,
  'Apollyon Pringle': 0.2558139534883721,
  'Cedric Diggory': 0.358695652173913,
  'Winky': 0.34789156626506024,
  'Andrew Kirke': 0.218957345971564,
}

```

Weighted Closeness Centrality

```

closeness_cent = nx.closeness centrality(HPC, distance='weight')
closeness_cent

```

```

{
  'Adalbert Waffling': 0.05965909090909091,
  'Bathilda Bagshot': 0.08950019372336304,
  'Miranda Goshawk': 0.06477846326416152,
  'Newt Scamander': 0.04408396946564885,
  'Adrian Pucey': 0.07293969055888853,
  'George Weasley': 0.10953058321479374,
  'Lee Jordan': 0.11797752808988764,
  'Marcus Flint': 0.08421436383521691,
  'Terence Higgs': 0.06740589436825212,
  'Agatha Timms': 0.06856634016028496,
  'Ludo Bagman': 0.08622620380739082,
  'Roddy Pontner': 0.06856634016028496,
  'Aidan Lynch': 0.04342921601804851,
  'Connolly': 0.04075511644318984,
  'Quigley': 0.05241660993873383,
  'Albus Dumbledore': 0.11100432484382508,
  'Augusta Longbottom': 0.09995672868887927,
  'Avery': 0.08514559528197567,
  'Billy Stubbs': 0.09090909090909091,
  'Charity Burbage': 0.0995260663507109,
  'Dobby': 0.10581768208886853,
  'Draco Malfoy': 0.11526946107784432,
  'Gellert Grindelwald': 0.059049079754601226,
  'Griselda Marchbanks': 0.08455344070278184,
}

```

```
'Harry Potter': 0.12018730489073881,
'Karkus': 0.06697593505363873,
'Mad-Eye Moody': 0.1056241426611797,
'Nicolas Flamel': 0.09258517034068137,
'Pansy Parkinson': 0.10679611650485436,
'Percy Weasley': 0.11702127659574468,
'Rita Skeeter': 0.09961190168175937,
'Rubeus Hagrid': 0.09705882352941177,
'Severus Snape': 0.11224489795918367,
'Sirius Black': 0.111218103033221,
'Tiberius Ogden': 0.07702567522507503,
'Xenophilius Lovegood': 0.10284951024042743,
'Alecto Carrow': 0.07337992376111817,
'Merlin': 0.07389635316698656,
'Alice Longbottom': 0.06766256590509666,
'Emmeline Vance': 0.09173947577442415,
'Marlene McKinnon': 0.05090348171000441,
'Alicia Spinnet': 0.08932714617169374,
'Angelina Johnson': 0.08217716115261472,
'Cho Chang': 0.09322033898305085,
'Colin Creevey': 0.10386690647482015,
'Geoffrey Hooper': 0.07583716349310571,
'Katie Bell': 0.08489525909592062,
'Luna Lovegood': 0.11846153846153847,
'Miles Bletchley': 0.060031185031185035,
'Warrington': 0.07736101808439384,
'Ambrosius Flume': 0.05811320754716981,
'Barnabas Cuffe': 0.08148148148148149,
'Gwenog Jones': 0.045073170731707315,
'Amos Diggory': 0.07940873152286008,
'Apollyon Pringle': 0.06418449569324812,
'Cedric Diggory': 0.09141274238227147,
'Winky': 0.06939020726945029,
'Andrew Kirke': 0.06278880130470237,
```

We need to find the node with the smallest closeness centrality:

```
min_node_closeness = min(closeness_cent, key=closeness_cent.get)
min_centrality_closeness = closeness_cent[min_node_closeness]
print("Node with the smallest closeness centrality:", min_node_closeness)
print("Smallest closeness centrality:", min_centrality_closeness)
```

```
Node with the smallest closeness centrality: Mundungus Fletcher
Smallest closeness centrality: 0.029611588257915652
```

We need to find the node with the largest closeness centrality:

```
max_node_closeness = max(closeness_cent, key=closeness_cent.get)
max_centrality_closeness = closeness_cent[max_node_closeness]
print("Node with the largest closeness centrality:", max_node_closeness)
print("Largest closeness centrality:", max_centrality_closeness)
```

```
Node with the largest closeness centrality: Harry Potter
Largest closeness centrality: 0.12018730489073881
```

Eigenvector Centrality

- Eigenvector centrality measures the importance of a node in the network by considering the centrality of its neighbors.
- Nodes with high eigenvector centrality are connected to other nodes that are themselves central, indicating their influence within the network. It considers both the number of connections a node has and the importance of the nodes it is connected to.
- In this case, if a node has a high eigenvector centrality value, it means that the given character surrounds him/herself with other important, influential characters.

```
eigenvector_cent = nx.eigenvector_centrality(HPC)
eigenvector_cent
```

```
{'Adalbert Waffling': 0.005015109758148338,
'Bathilda Bagshot': 0.05409388347338553,
'Miranda Goshawk': 0.006846243128681326,
'Newt Scamander': 0.0004126960903932975,
'Adrian Pucey': 0.024327250501763816,
'George Weasley': 0.11481534428020206,
'Lee Jordan': 0.1379727994610037,
'Marcus Flint': 0.039572732346539956,
'Terence Higgs': 0.005224012414612291,
'Agatha Timms': 0.009491491345784841,
'Ludo Bagman': 0.1066232351422215,
'Roddy Pontner': 0.009491491345784841,
'Aidan Lynch': 7.534862578067618e-05,
'Connolly': 9.646117666916498e-05,
'Quigley': 0.0008253375306085725,
'Albus Dumbledore': 0.1609695798609104,
```



```
'Augusta Longbottom': 0.013157987596108346,
'Avery': 0.023580108629193967,
'Billy Stubbs': 0.013157987596108346,
'Charity Burbage': 0.0318767801175644,
'Dobby': 0.2317014097301836,
'Draco Malfoy': 0.21672302884383987,
'Gellert Grindelwald': 0.017579731826739782,
'Griselda Marchbanks': 0.032164025000166185,
'Harry Potter': 0.4309495376824086,
'Karkus': 0.013246492845237271,
'Mad-Eye Moody': 0.13496728273730615,
'Nicolas Flamel': 0.027057307092068515,
'Pansy Parkinson': 0.12275166385565754,
'Percy Weasley': 0.11324430567232692,
'Rita Skeeter': 0.1625784423873903,
'Rubeus Hagrid': 0.05397565327381666,
'Severus Snape': 0.12142156336681273,
'Sirius Black': 0.13751636985295998,
'Tiberius Ogden': 0.01578712777952691,
'Xenophilius Lovegood': 0.055785319520857676,
'Alecto Carrow': 0.011764545249227982,
'Merlin': 0.02250158082202116,
'Alice Longbottom': 0.0021829113918367567,
'Emmeline Vance': 0.02647269261252301,
'Marlene McKinnon': 0.00023255366357739313,
'Alicia Spinnet': 0.07506495843825936,
'Angelina Johnson': 0.04925691215415182,
'Cho Chang': 0.07305007234034334,
'Colin Creevey': 0.14564673534966252,
'Geoffrey Hooper': 0.006136563746551194,
'Katie Bell': 0.11300307586339375,
'Luna Lovegood': 0.16763117774784667,
'Miles Bletchley': 0.006612206783357786,
'Warrington': 0.06455679741002848,
'Ambrosius Flume': 0.00016626695070152137,
'Barnabas Cuffe': 0.002020475191617429,
'Gwenog Jones': 1.359057542167745e-05,
'Amos Diggory': 0.064496939597692,
'Apollyon Pringle': 0.005272109274654503,
'Cedric Diggory': 0.09181928563970707,
'Winky': 0.07590322133390963,
'Andrew Kirke': 0.0011504561947577887,
```

Weighted Eigenvector Centrality

```
eigenvector_cent = nx.eigenvector_centrality_numpy(HPC, weight='weight')
eigenvector_cent
```

```
{'Adalbert Waffling': 1.86317591721079e-05,
'Bathilda Bagshot': 0.0028720826007751265,
'Miranda Goshawk': 1.9331878976475818e-05,
'Newt Scamander': 1.201936675853079e-07,
'Adrian Pucey': 5.6109583759814794e-05,
'George Weasley': 0.001482801372648468,
'Lee Jordan': 0.00717301568956364,
'Marcus Flint': 4.218445504534544e-05,
'Terence Higgs': 3.170347281229274e-07,
'Agatha Timms': 2.765090758751923e-05,
'Ludo Bagman': 0.00851763717964409,
'Roddy Pontner': 2.7650907587528512e-05,
'Aidan Lynch': 6.84698961753569e-10,
'Connolly': 6.22891353601979e-10,
'Quigley': 1.582796766819554e-07,
'Albus Dumbledore': 0.03249386929832316,
'Augusta Longbottom': 3.493492634873282e-05,
'Avery': 0.0001286195643648885,
'Billy Stubbs': 6.986985269749114e-05,
'Charity Burbage': 0.00015169823838695595,
'Dobby': 0.6906904598163915,
'Draco Malfoy': 0.02162235299470765,
'Gellert Grindelwald': 0.00040281125287157653,
'Griselda Marchbanks': 0.00023363766188229217,
'Harry Potter': 0.6600656703194637,
'Karkus': 0.0002096098003785238,
'Mad-Eye Moody': 0.006166665627235773,
'Nicolas Flamel': 0.00014913775503428995,
'Pansy Parkinson': 0.005200287975502131,
'Percy Weasley': 0.003254650211860956,
'Rita Skeeter': 0.014217869524902229,
'Rubeus Hagrid': 0.0004555497268677167,
'Severus Snape': 0.003514309085620447,
'Sirius Black': 0.01039577994310356,
'Tiberius Ogden': 0.0001412468416061653,
'Xenophilius Lovegood': 0.006428098533380432,
'Alecto Carrow': 1.9546044188588524e-05,
'Merlin': 0.00020291021070347864,
'Alice Longbottom': 3.707505609303313e-07,
'Emmeline Vance': 8.620808238044841e-05,
```

```
'Marlene McKinnon': 2.451609098224181e-09,
'Alicia Spinnet': 0.00014494973707377252,
'Angelina Johnson': 0.0001035571102260792,
'Cho Chang': 0.00016208901814877027,
'Colin Creevey': 0.0056676013479044956,
'Geoffrey Hooper': 3.116777717362638e-07,
'Katie Bell': 0.0030296268160600833,
'Luna Lovegood': 0.008419838727528717,
'Miles Bletchley': 9.363520658565355e-07,
'Warrington': 0.00018952502237321083,
'Ambrosius Flume': 9.797530934377822e-10,
'Barnabas Cuffe': 1.8225341804018961e-07,
'Gwenog Jones': 5.266780168870118e-12,
'Amos Diggory': 0.009810904788092269,
'Apollyon Pringle': 3.164380637837496e-05,
'Cedric Diggory': 0.004891644110832597,
'Winky': 0.24008788033503514,
'Andrew Kinko': 0.7201502607021020e-00
```

We need to find the node with the smallest eigenvector centrality:

```
min_node_eigenvector = min(eigenvector_cent, key=eigenvector_cent.get)
min_centrality_eigenvector = eigenvector_cent[min_node_eigenvector]
print("Node with the smallest eigenvector centrality:", min_node_eigenvector)
print("Smallest eigenvector centrality:", min_centrality_eigenvector)
```

```
Node with the smallest eigenvector centrality: Gwenog Jones
Smallest eigenvector centrality: 5.266780168870118e-12
```

We need to find the node with the largest eigenvector centrality:

```
max_node_eigenvector = max(eigenvector_cent, key=eigenvector_cent.get)
max_centrality_eigenvector = eigenvector_cent[max_node_eigenvector]
print("Node with the largest eigenvector centrality:", max_node_eigenvector)
print("Largest eigenvector centrality:", max_centrality_eigenvector)
```

```
Node with the largest eigenvector centrality: Dobby
Largest eigenvector centrality: 0.6906904598163915
```

PageRank Centrality

- PageRank evaluates the importance of a node in a network by considering both the quantity and quality of its incoming links. It assigns higher scores to nodes that are linked to by other nodes with high PageRank scores, recursively propagating importance throughout the network.
- Nodes with high PageRank are not only well-connected but also connected to other important nodes, making them pivotal in information dissemination or authority distribution within the network. In web networks, nodes with high PageRank often correspond to web pages with high visibility or credibility.
- In our case, pagerank centrality measures the all in all importance and influence and trustworthiness of a character (node).

```
pagerank_cent = nx.pagerank(HPC)
pagerank_cent
```

```
{'Adalbert Waffling': 0.0039127311626514045,
'Bathilda Bagshot': 0.008024876357337429,
'Miranda Goshawk': 0.003817662263673451,
'Newt Scamander': 0.003708754899145449,
'Adrian Pucey': 0.002753458075614708,
'George Weasley': 0.007658111197066391,
'Lee Jordan': 0.010074725218418132,
'Marcus Flint': 0.003995686978505953,
'Terence Higgs': 0.0012806049073199354,
'Agatha Timms': 0.0020596055199212544,
'Ludo Bagman': 0.008970257430430338,
'Roddy Pontner': 0.0020596055199212544,
'Aidan Lynch': 0.0026710471313265804,
'Connolly': 0.002742861512040452,
'Quigley': 0.003097949623318781,
'Albus Dumbledore': 0.014011988962649794,
'Augusta Longbottom': 0.0007316257203673266,
'Avery': 0.007060482080117742,
'Billy Stubbs': 0.0008166997165967222,
'Charity Burbage': 0.0009425208775117792,
'Dobby': 0.07241599651024082,
'Draco Malfoy': 0.013102328663387475,
'Gellert Grindelwald': 0.002313526430894027,
'Griselda Marchbanks': 0.0021998084288417166,
'Harry Potter': 0.07929411987153934,
'Karkus': 0.0019029766728118,
'Mad-Eye Moody': 0.004644029840069629,
'Nicolas Flamel': 0.0021593755780367307,
'Pansy Parkinson': 0.003725738866695962,
```

```
'Percy Weasley': 0.004977701570467628,
'Rita Skeeter': 0.01087145254079998,
'Rubeus Hagrid': 0.0018695943516157487,
'Severus Snape': 0.004279170560944288,
'Sirius Black': 0.011207457762053964,
'Tiberius Ogden': 0.0017881892815777593,
'Xenophilius Lovegood': 0.0029391432877657487,
'Alecto Carrow': 0.0015028476771196137,
'Merlin': 0.007048741064025624,
'Alice Longbottom': 0.0025974767512992867,
'Emmeline Vance': 0.005658389131131224,
'Marlene McKinnon': 0.002862245799126307,
'Alicia Spinnet': 0.006906440850634049,
'Angelina Johnson': 0.00511680524471986,
'Cho Chang': 0.004681363156451245,
'Colin Creevey': 0.004884061740807921,
'Geoffrey Hooper': 0.0008300105003078431,
'Katie Bell': 0.00835360375009907,
'Luna Lovegood': 0.005691924351795816,
'Miles Bletchley': 0.0016158382098778153,
'Warrington': 0.008830258584337407,
'Ambrosius Flume': 0.004249832736226703,
'Barnabas Cuffe': 0.002863880524939747,
'Gwenog Jones': 0.002458961159535849,
'Amos Diggory': 0.004235896712583253,
'Apollyon Pringle': 0.000955140904859787,
'Cedric Diggory': 0.004673045610475004,
'Winky': 0.018235184172040203,
'Andrew Kirke': 0.00250327043011345
```

Weighted Pagerank Centrality

```
pagerank_cent = nx.pagerank(HPC, weight='weight')
pagerank_cent
```

```
{'Adalbert Waffling': 0.0039127311626514045,
'Bathilda Bagshot': 0.008024876357337429,
'Miranda Goshawk': 0.003817662263673451,
'Newt Scamander': 0.003708754899145449,
'Adrian Pucey': 0.002753458075614708,
'George Weasley': 0.007658111197066391,
'Lee Jordan': 0.010074725218418132,
'Marcus Flint': 0.003995686978505953,
'Terence Higgs': 0.0012806049073199354,
'Agatha Timms': 0.0020596055199212544,
'Ludo Bagman': 0.008970257430430338,
'Roddy Pontner': 0.0020596055199212544,
'Aidan Lynch': 0.0026710471313265804,
'Connolly': 0.002742861512040452,
'Quigley': 0.003097949623318781,
'Albus Dumbledore': 0.014011988962649794,
'Augusta Longbottom': 0.0007316257203673266,
'Avery': 0.007060482080117742,
'Billy Stubbs': 0.0008166997165967222,
'Charity Burbage': 0.0009425208775117792,
'Dobby': 0.07241599651024082,
'Draco Malfoy': 0.013102328663387475,
'Gellert Grindelwald': 0.002313526430894027,
'Griselda Marchbanks': 0.0021998084288417166,
'Harry Potter': 0.07929411987153934,
'Karkus': 0.0019029766728118,
'Mad-Eye Moody': 0.004644029840069629,
'Nicolas Flamel': 0.0021593755780367307,
'Pansy Parkinson': 0.003725738866695962,
'Percy Weasley': 0.004977701570467628,
'Rita Skeeter': 0.01087145254079998,
'Rubeus Hagrid': 0.0018695943516157487,
'Severus Snape': 0.004279170560944288,
'Sirius Black': 0.011207457762053964,
'Tiberius Ogden': 0.0017881892815777593,
'Xenophilius Lovegood': 0.0029391432877657487,
'Alecto Carrow': 0.0015028476771196137,
'Merlin': 0.007048741064025624,
'Alice Longbottom': 0.0025974767512992867,
'Emmeline Vance': 0.005658389131131224,
'Marlene McKinnon': 0.002862245799126307,
'Alicia Spinnet': 0.006906440850634049,
'Angelina Johnson': 0.00511680524471986,
'Cho Chang': 0.004681363156451245,
'Colin Creevey': 0.004884061740807921,
'Geoffrey Hooper': 0.0008300105003078431,
'Katie Bell': 0.00835360375009907,
'Luna Lovegood': 0.005691924351795816,
'Miles Bletchley': 0.0016158382098778153,
'Warrington': 0.008830258584337407,
'Ambrosius Flume': 0.004249832736226703,
'Barnabas Cuffe': 0.002863880524939747,
'Gwenog Jones': 0.002458961159535849,
'Amos Diggory': 0.004235896712583253,
```

```
'Apollyon Pringle': 0.000955140904859787,
'Cedric Diggory': 0.004673045610475004,
'Winky': 0.018235184172040203,
'Andrew Kirke': 0.00259327943911345,
```

We need to find the node with the smallest pagerank centrality:

```
min_node_pagerank = min(pagerank_cent, key=pagerank_cent.get)
min_centrality_pagerank = pagerank_cent[min_node_pagerank]
print("Node with the smallest pagerank centrality:", min_node_pagerank)
print("Smallest pagerank centrality:", min_centrality_pagerank)
```

```
Node with the smallest pagerank centrality: Marvolo Gaunt
Smallest pagerank centrality: 0.0006987642575799543
```

We need to find the node with the largest pagerank centrality:

```
max_node_pagerank = max(pagerank_cent, key=pagerank_cent.get)
max_centrality_pagerank = pagerank_cent[max_node_pagerank]
print("Node with the largest pagerank centrality:", max_node_pagerank)
print("Largest pagerank centrality:", max_centrality_pagerank)
```

```
Node with the largest pagerank centrality: Harry Potter
Largest pagerank centrality: 0.07929411987153934
```

Harmonic Centrality

- Harmonic centrality measures the centrality of a node in a network by considering the sum of the reciprocals of the shortest path distances from that node to all other nodes. It emphasizes nodes that are close to many other nodes in the network, assigning higher scores to nodes that are not only well-connected but also have short paths to other nodes.
- Nodes with high harmonic centrality are positioned to efficiently communicate or exchange information with other nodes, playing crucial roles in facilitating interactions or spreading influence within the network.
- In the Harry Potter network, if a node has a high harmonic centrality value, it means that the given character has a very central role, information flows through him/her, others would not be able to effectively communicate without him/her.

```
harmonic_cent = nx.harmonic centrality(HPC)
harmonic_cent
```

```
{'Peter Pettigrew': 73.89285714285721,
'Aragog': 77.04285714285722,
'Padma Patil': 74.66666666666671,
'Bogrod': 84.14999999999995,
'Justin Finch-Fletchley': 87.98333333333325,
'Ginny Weasley': 94.81666666666656,
'Ernie Macmillan': 92.23333333333326,
'Fleur Delacour': 81.6166666666665,
'Cormac McLaggen': 78.89999999999999,
'Tobias Snape': 68.20952380952392,
'Archie': 70.73333333333346,
'Madam Rosmerta': 70.29285714285723,
'Alicia Spinnet': 84.2166666666665,
'Lucius Malfoy': 98.61666666666655,
'Connolly': 45.84444444444434,
'Apollyon Pringle': 62.51904761904773,
'Oliver Wood': 86.02619047619042,
'Auntie Muriel': 88.61666666666656,
'Crookshanks': 84.94999999999992,
'Tiberius Ogden': 70.15952380952392,
'Hedwig': 107.74999999999983,
'Bellatrix Lestrange': 90.53333333333327,
'Fang': 79.89285714285717,
'Nagini': 87.14999999999992,
'Marvolo Gaunt': 69.82619047619056,
'Bertha Jorkins': 71.5095238095239,
'Griselda Marchbanks': 76.28333333333335,
'Roddy Pontner': 66.23333333333345,
'Bletchley': 78.06666666666667,
'Edgar Bones': 63.50238095238105,
'Zacharias Smith': 87.13333333333328,
'Anthony Goldstein': 79.80238095238099,
'Zograf': 38.782539682539735,
'Ambrosius Flume': 47.265476190476214,
'Armando Dippet': 72.13333333333344,
'Colin Creevey': 97.53333333333325,
'Madam Puddifoot': 54.7392857142858,
'Miranda Goshawk': 67.27619047619055,
'Ernie Prang': 74.22619047619051,
'Hermes': 73.05952380952387,
'Agatha Timms': 66.23333333333345,
'Severus Snape': 96.39999999999985,
'Dennis Creevey': 74.77619047619056,
```

```
'Dolores Umbridge': 82.84999999999995,
'Sybill Trelawney': 64.73571428571438,
'Madam Malkin': 72.80000000000004,
'George Weasley': 96.816666666666649,
'Troy': 57.67976190476198,
'Billy Stubbs': 69.6595238095239,
'Demelza Robins': 72.93571428571438,
'Bane': 69.00238095238106,
'Scabbers': 77.24285714285719,
'Seamus Finnigan': 83.94999999999999,
'Fawkes': 89.23333333333326,
'Ritchie Coote': 51.26666666666673,
'Frank Bryce': 83.14999999999996,
'Kendra Dumbledore': 83.14999999999995,
'Eldred Worple': 60.28571428571438,
```

Weighted Harmonic Centrality

```
harmonic_cent = nx.harmonic centrality(HPC, distance='weight')
harmonic_cent
```

```
{'Peter Pettigrew': 16.918203561287914,
'Aragog': 19.854500281728704,
'Padma Patil': 28.08142384678529,
'Bogrod': 18.902858755277755,
'Justin Finch-Fletchley': 19.53770005066892,
'Ginny Weasley': 31.709385046735065,
'Ernie Macmillan': 29.39397477006996,
'Fleur Delacour': 29.910701566173344,
'Cormac McLaggen': 23.25191147992165,
'Tobias Snape': 15.29857529470876,
'Archie': 20.609981014427607,
'Madam Rosmerta': 22.041330631891103,
'Alicia Spinnet': 24.005995732012856,
'Lucius Malfoy': 23.76260516014772,
'Connolly': 10.033301885356794,
'Apollyon Pringle': 16.46944749757848,
'Oliver Wood': 29.388888154217696,
'Auntie Muriel': 26.49234255238514,
'Crookshanks': 29.402917945724603,
'Tiberius Ogden': 20.02539039264977,
'Hedwig': 34.65746731189426,
'Bellatrix Lestrange': 28.085607174473267,
'Fang': 25.066542188928626,
'Nagini': 22.42327334303274,
'Marvolo Gaunt': 23.123354509143304,
'Bertha Jorkins': 26.643187395281647,
'Griselda Marchbanks': 22.324814749695364,
'Roddy Pontner': 17.331634604745048,
'Bletchley': 31.506503933885824,
'Edgar Bones': 17.856682971348818,
'Zacharias Smith': 30.112368370957356,
'Anthony Goldstein': 22.104663958413642,
'Zograf': 8.755083196042914,
'Ambrosius Flume': 14.437942568778976,
'Armando Dippet': 21.035805408860014,
'Colin Creevey': 28.85973779254445,
'Madam Puddifoot': 15.268381615451545,
'Miranda Goshawk': 16.076817467990313,
'Ernie Prang': 28.78408045336802,
'Hermes': 26.412038514685246,
'Agatha Timms': 17.331634604745048,
'Severus Snape': 32.52486551932759,
'Dennis Creevey': 21.34298029117295,
'Dolores Umbridge': 22.329654580794475,
'Sybill Trelawney': 9.39673447981549,
'Madam Malkin': 18.508392964777407,
'George Weasley': 31.746295340449677,
'Troy': 12.842698804961035,
'Billy Stubbs': 25.032664881800855,
'Demelza Robins': 18.220542018775003,
'Bane': 16.40602045141697,
'Scabbers': 30.73603159872181,
'Seamus Finnigan': 26.836372840490167,
'Fawkes': 24.81911515321698,
'Ritchie Coote': 13.316296581139172,
'Frank Bryce': 23.263015445250286,
'Kendra Dumbledore': 20.63062650195028,
'Eldred Worple': 12.602894132485329,
```

We need to find the node with the smallest harmonic centrality:

```
min_node_harmonic = min(harmonic_cent, key=harmonic_cent.get)
min_centrality_harmonic = harmonic_cent[min_node_harmonic]
print("Node with the smallest harmonic centrality:", min_node_harmonic)
print("Smallest harmonic centrality:", min_centrality_harmonic)
```

```
Node with the smallest harmonic centrality: Mundungus Fletcher
Smallest harmonic centrality: 6.943703998415049
```

We need to find the node with the largest harmonic centrality:

```
max_node_harmonic = max(harmonic_cent, key=harmonic_cent.get)
max_centrality_harmonic = harmonic_cent[max_node_harmonic]
print("Node with the largest harmonic centrality:", max_node_harmonic)
print("Largest harmonic centrality:", max_centrality_harmonic)
```

```
Node with the largest harmonic centrality: Percy Weasley
Largest harmonic centrality: 37.016615342226494
```

Load Centrality

- Load centrality evaluates the centrality of a node in a network based on the amount of traffic or load that passes through it. Nodes that lie on many short paths between other nodes tend to have high load centrality, indicating their importance in facilitating communication or resource exchange within the network.
- Nodes with high load centrality serve as critical junctions for information or resource flow, playing key roles in maintaining network connectivity and efficiency.
- In this case, if a node has a high load centrality value, it means that many others keep in touch, communicate through the given character.

```
load_cent = nx.load_centrality(HPC)
load_cent
```

```
{'Adalbert Waffling': 0.01836062488236401,
 'Bathilda Bagshot': 0.052471477598871545,
 'Miranda Goshawk': 0.010760062918448011,
 'Newt Scamander': 0.008658008658008656,
 'Adrian Pucey': 0.0037313743105139197,
 'George Weasley': 0.022119403382832383,
 'Lee Jordan': 0.06483969711771805,
 'Marcus Flint': 0.0057311612813036195,
 'Terence Higgs': 0.0,
 'Agatha Timms': 0.0,
 'Ludo Bagman': 0.06144180703983185,
 'Roddy Pontner': 0.0,
 'Aidan Lynch': 0.0002604656816613338,
 'Connolly': 0.0006551065968767832,
 'Quigley': 0.008517755565115814,
 'Albus Dumbledore': 0.08045597340869151,
 'Augusta Longbottom': 0.0,
 'Avery': 0.022860410523454,
 'Billy Stubbs': 0.0,
 'Charity Burbage': 0.00041762997469519216,
 'Dobby': 0.032879489402219464,
 'Draco Malfoy': 0.10527448034316798,
 'Gellert Grindelwald': 0.0007247609188602976,
 'Griselda Marchbanks': 0.0008046917713066781,
 'Harry Potter': 0.39942598957098513,
 'Karkus': 0.008658008658008658,
 'Mad-Eye Moody': 0.021282671516890634,
 'Nicolas Flamel': 0.012666746833413506,
 'Pansy Parkinson': 0.024063109835180296,
 'Percy Weasley': 0.04769796362608297,
 'Rita Skeeter': 0.03439191371808303,
 'Rubeus Hagrid': 0.000423185194147927,
 'Severus Snape': 0.013469911298429393,
 'Sirius Black': 0.08688041032960613,
 'Tiberius Ogden': 0.0,
 'Xenophilius Lovegood': 0.014530221929937269,
 'Alecto Carrow': 0.0007273825208607816,
 'Merlin': 0.026768418683532556,
 'Alice Longbottom': 0.007390664275651571,
 'Emmeline Vance': 0.02470851233354759,
 'Marlene McKinnon': 0.0005552418595896856,
 'Alicia Spinnet': 0.018930610429716388,
 'Angelina Johnson': 0.0012910240652335516,
 'Cho Chang': 0.008414203032991851,
 'Colin Creevey': 0.016422029447450588,
 'Geoffrey Hooper': 0.0,
 'Katie Bell': 0.009555813838205753,
 'Luna Lovegood': 0.017042227011824833,
 'Miles Bletchley': 0.0004208427442123094,
 'Warrington': 0.017153490751446242,
 'Ambrosius Flume': 0.008658008658008658,
 'Barnabas Cuffe': 0.017240730284208545,
 'Gwenog Jones': 0.0,
 'Amos Diggory': 0.015948934529162285,
 'Apollyon Pringle': 0.0,
 'Cedric Diggory': 0.008810338656579597,
 'Winky': 0.004360125333986622,
 'Andrew Kirke': 0.0,
```

Weighted Load Centrality

```
load_cent = nx.load_centrality(HPC, weight='weight')
load_cent
```

```
{'Adalbert Waffling': 0.017730095990965554,
 'Bathilda Bagshot': 0.04607644770688249,
 'Miranda Goshawk': 0.013226990400903442,
 'Newt Scamander': 0.008658008658008658,
 'Adrian Pucey': 0.0,
 'George Weasley': 0.037146023978322104,
 'Lee Jordan': 0.14373649987004014,
 'Marcus Flint': 0.009185017880670053,
 'Terence Higgs': 3.1369596586987884e-05,
 'Agatha Timms': 0.0,
 'Ludo Bagman': 0.05045015371102328,
 'Roddy Pontner': 0.0,
 'Aidan Lynch': 0.0004705439488048183,
 'Connolly': 0.0,
 'Quigley': 0.009308927787188656,
 'Albus Dumbledore': 0.1109879854445071,
 'Augusta Longbottom': 0.0,
 'Avery': 0.01941150636802811,
 'Billy Stubbs': 0.0,
 'Charity Burbage': 0.016235334713595586,
 'Dobby': 0.04765423384988602,
 'Draco Malfoy': 0.14779435138130786,
 'Gellert Grindelwald': 0.0,
 'Griselda Marchbanks': 0.0,
 'Harry Potter': 0.2909611644394255,
 'Karkus': 0.008658008658008658,
 'Mad-Eye Moody': 0.045783926218708824,
 'Nicolas Flamel': 0.02067413263065437,
 'Pansy Parkinson': 0.055012357380369796,
 'Percy Weasley': 0.15523349436392916,
 'Rita Skeeter': 0.04890088775958341,
 'Rubeus Hagrid': 0.005235585670368278,
 'Severus Snape': 0.06966863249471941,
 'Sirius Black': 0.10453719388502002,
 'Tiberius Ogden': 0.0,
 'Xenophilius Lovegood': 0.0180202647593952,
 'Alecto Carrow': 0.0010634293242988895,
 'Merlin': 0.01990557751427317,
 'Alice Longbottom': 0.008557625948930296,
 'Emmeline Vance': 0.0253842775581906,
 'Marlene McKinnon': 0.00015057406361754187,
 'Alicia Spinnet': 0.014850994416211809,
 'Angelina Johnson': 7.528703180877094e-05,
 'Cho Chang': 0.008462038306758787,
 'Colin Creevey': 0.014627172344563646,
 'Geoffrey Hooper': 0.0,
 'Katie Bell': 0.0015857331074722375,
 'Luna Lovegood': 0.12061053450805011,
 'Miles Bletchley': 0.00045015371102327624,
 'Warrington': 0.011211493820189473,
 'Ambrosius Flume': 0.008658008658008658,
 'Barnabas Cuffe': 0.017240730284208545,
 'Gwenog Jones': 0.0,
 'Amos Diggory': 0.01680782985130811,
 'Apollyon Pringle': 0.0,
 'Cedric Diggory': 0.010017880670054585,
 'Winky': 0.0004674069891461196,
 'Andrew Kirke': 0.0,
```

We need to find the node with the smallest load centrality:

```
min_node_load = min(load_cent, key=load_cent.get)
min_centrality_load = load_cent[min_node_load]
print("Node with the smallest load centrality:", min_node_load)
print("Smallest load centrality:", min_centrality_load)
```

```
Node with the smallest load centrality: Adrian Pucey
Smallest load centrality: 0.0
```

We need to find the node with the largest load centrality:

```
max_node_load = max(load_cent, key=load_cent.get)
max_centrality_load = load_cent[max_node_load]
print("Node with the largest load centrality:", max_node_load)
print("Largest load centrality:", max_centrality_load)
```

```
Node with the largest load centrality: Harry Potter
Largest load centrality: 0.2909611644394255
```

Now, we can **visualize** the comparison between these centralities in every combination:

```
from itertools import combinations

centrality_measures = {
    'degree': degree_cent,
    'between': betweenness_cent,
    'closeness': closeness_cent,
    'eigenvector': eigenvector_cent,
    'pagerank': pagerank_cent,
    'harmonic': harmonic_cent,
    'load': load_cent
}

centrality_colors = {
    'between': griff_colors[0],
    'degree': griff_colors[2],
    'closeness': slyth_colors[0],
    'eigenvector': slyth_colors[2],
    'pagerank': huff_colors[0],
    'harmonic': huff_colors[2],
    'load': raven_colors[3]
}

nodes = list(degree_cent.keys())
num_measures = len(centrality_measures)

fig, axs = plt.subplots(5, 5, figsize=(15, 15))

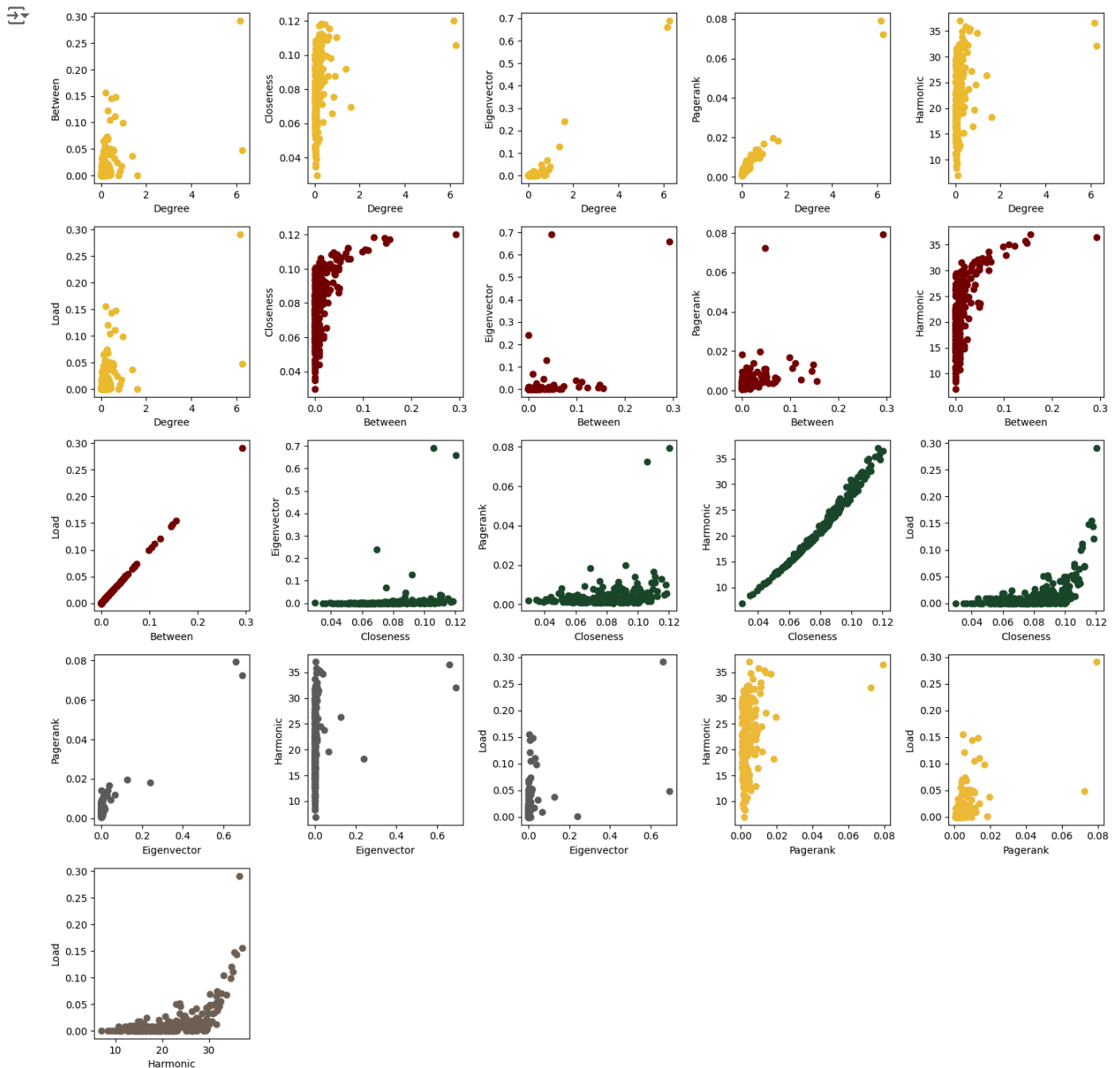
comb = list(combinations(centrality_measures.keys(), 2))
for i, (centrality1, centrality2) in enumerate(comb):
    row = i // 5
    col = i % 5
    ax = axs[row, col]

    ax.scatter(
        [centrality_measures[centrality1][node] for node in nodes],
        [centrality_measures[centrality2][node] for node in nodes],
        color=centrality_colors[centrality1]
    )

    ax.set_xlabel(centrality1.capitalize())
    ax.set_ylabel(centrality2.capitalize())

for i in range(len(comb), 25):
    axs[i // 5, i % 5].axis('off')

plt.tight_layout()
plt.show()
```

Here we can look for **correlations** between the respective measures.

Identifying the top 10 nodes for each metric

```

top_degree_nodes = sorted(degree_cent, key=degree_cent.get, reverse=True)[:10]
top_betweenness_nodes = sorted(betweenness_cent, key=betweenness_cent.get, reverse=True)[:10]
top_closeness_nodes = sorted(closeness_cent, key=closeness_cent.get, reverse=True)[:10]
top_eigenvector_nodes = sorted(eigenvector_cent, key=eigenvector_cent.get, reverse=True)[:10]
top_pagerank_nodes = sorted(pagerank_cent, key=pagerank_cent.get, reverse=True)[:10]
top_harmonic_nodes = sorted(harmonic_cent, key=harmonic_cent.get, reverse=True)[:10]
top_load_nodes = sorted(load_cent, key=load_cent.get, reverse=True)[:10]

```

```

print("Top 10 nodes based on Degree Centrality:")
print(top_degree_nodes)
print("\nTop 10 nodes based on Betweenness Centrality:")
print(top_betweenness_nodes)
print("\nTop 10 nodes based on Closeness Centrality:")
print(top_closeness_nodes)
print("\nTop 10 nodes based on Eigenvector Centrality:")
print(top_eigenvector_nodes)
print("\nTop 10 nodes based on Pagerank Centrality:")
print(top_pagerank_nodes)
print("\nTop 10 nodes based on Harmonic Centrality:")
print(top_harmonic_nodes)
print("\nTop 10 nodes based on Load Centrality:")
print(top_load_nodes)

```

```

Top 10 nodes based on Degree Centrality:
['Dobby', 'Harry Potter', 'Winky', 'Kreacher', 'Hedwig', 'Firenze', 'Griphook', 'Bane', 'Dean Thomas', 'Draco Malfoy']

Top 10 nodes based on Betweenness Centrality:
['Harry Potter', 'Percy Weasley', 'Draco Malfoy', 'Lee Jordan', 'Luna Lovegood', 'Albus Dumbledore', 'Sirius Black', 'Hedwig', 'Ginr

Top 10 nodes based on Closeness Centrality:
['Harry Potter', 'Luna Lovegood', 'Lee Jordan', 'Percy Weasley', 'Draco Malfoy', 'Neville Longbottom', 'Severus Snape', 'Sirius Blac

Top 10 nodes based on Eigenvector Centrality:
['Dobby', 'Harry Potter', 'Winky', 'Kreacher', 'Griphook', 'Lucius Malfoy', 'Hedwig', 'Albus Dumbledore', 'Firenze', 'Draco Malfoy'

Top 10 nodes based on Pagerank Centrality:
['Harry Potter', 'Dobby', 'Kreacher', 'Winky', 'Hedwig', 'Dean Thomas', 'Albus Dumbledore', 'Draco Malfoy', 'Griphook', 'Firenze']

Top 10 nodes based on Harmonic Centrality:
['Percy Weasley', 'Harry Potter', 'Lee Jordan', 'Draco Malfoy', 'Albus Dumbledore', 'Luna Lovegood', 'Hedwig', 'Neville Longbottom',

Top 10 nodes based on Load Centrality:
['Harry Potter', 'Percy Weasley', 'Draco Malfoy', 'Lee Jordan', 'Luna Lovegood', 'Albus Dumbledore', 'Sirius Black', 'Hedwig', 'Ginr

```

We need to create a new dataframe to store these important nodes based on the centralities.

```

import pandas as pd

cent = {
    'Degree Centrality': top_degree_nodes,
    'Betweenness Centrality': top_betweenness_nodes,
    'Closeness Centrality': top_closeness_nodes,
    'Eigenvector Centrality': top_eigenvector_nodes,
    'Pagerank Centrality': top_pagerank_nodes,
    'Harmonic Centrality': top_harmonic_nodes,
    'Load Centrality': top_load_nodes,
}

important_nodes = pd.DataFrame(cent)

important_nodes.index += 1
important_nodes.index.name = 'Index'

important_nodes.to_csv('top_nodes centrality_hp.csv')

important_nodes

```



	Degree Centrality	Betweenness Centrality	Closeness Centrality	Eigenvector Centrality	Pagerank Centrality	Harmonic Centrality	Load Centrality
Index							
1	Dobby	Harry Potter	Harry Potter	Dobby	Harry Potter	Percy Weasley	Harry Potter
2	Harry Potter	Percy Weasley	Luna Lovegood	Harry Potter	Dobby	Harry Potter	Percy Weasley
3	Winky	Draco Malfoy	Lee Jordan	Winky	Kreacher	Lee Jordan	Draco Malfoy
4	Kreacher	Lee Jordan	Percy Weasley	Kreacher	Winky	Draco Malfoy	Lee Jordan
5	Hedwig	Luna Lovegood	Draco Malfoy	Griphook	Hedwig	Albus Dumbledore	Luna Lovegood
6	Firenze	Albus Dumbledore	Neville Longbottom	Lucius Malfoy	Dean Thomas	Luna Lovegood	Albus Dumbledore
7	Griphook	Sirius Black	Severus Snape	Hedwig	Albus Dumbledore	Hedwig	Sirius Black
8	Bane	Hedwig	Sirius Black	Albus Dumbledore	Draco Malfoy	Neville Longbottom	Hedwig
9	Dean Thomas	Ginny Weasley	Albus Dumbledore	Firenze	Griphook	Sirius Black	Ginny Weasley

When it comes to the top 10 nodes in our Harry Potter network, almost all centrality measures agree on the **Harry Potter** node to be **the most influential node**, which is not surprising, considering he is the main character of the books.

Dobby and **Percy Weasley** could be considered all in all the **second most influential** nodes, since they take the first or second place mostly at every centrality measure.

We find it surprising that **Hermione** and **Ron** do not really appear as influential nodes.

Now, we can **visualize the network** based on different centrality measures.

On these plots, the nodes are **colored** based on the value of the measures.

```
fig, axs = plt.subplots(3, 3, figsize=(18, 15))

nx.draw(HPC, pos=nx.spring_layout(HPC), ax=axs[0, 0], node_color=list(degree_cent.values()), cmap=plt.cm.OrRd, node_size=100)
axs[0, 0].set_title('Network of Sister Cities - Nodes Colored by Degree Centrality')

nx.draw(HPC, pos=nx.spring_layout(HPC), ax=axs[0, 1], node_color=list(betweenness_cent.values()), cmap=plt.cm.OrRd, node_size=100)
axs[0, 1].set_title('Network of Sister Cities - Nodes Colored by Betweenness Centrality')

nx.draw(HPC, pos=nx.spring_layout(HPC), ax=axs[0, 2], node_color=list(closeness_cent.values()), cmap=plt.cm.OrRd, node_size=100)
axs[0, 2].set_title('Network of Sister Cities - Nodes Colored by Closeness Centrality')

nx.draw(HPC, pos=nx.spring_layout(HPC), ax=axs[1, 0], node_color=list(eigenvector_cent.values()), cmap=plt.cm.OrRd, node_size=100)
axs[1, 0].set_title('Network of Sister Cities - Nodes Colored by Eigenvector Centrality')

nx.draw(HPC, pos=nx.spring_layout(HPC), ax=axs[1, 1], node_color=list(pagerank_cent.values()), cmap=plt.cm.OrRd, node_size=100)
axs[1, 1].set_title('Network of Sister Cities - Nodes Colored by Pagerank Centrality')

nx.draw(HPC, pos=nx.spring_layout(HPC), ax=axs[1, 2], node_color=list(harmonic_cent.values()), cmap=plt.cm.OrRd, node_size=100)
axs[1, 2].set_title('Network of Sister Cities - Nodes Colored by Harmonic Centrality')

nx.draw(HPC, pos=nx.spring_layout(HPC), ax=axs[2, 0], node_color=list(load_cent.values()), cmap=plt.cm.OrRd, node_size=100)
axs[2, 0].set_title('Network of Sister Cities - Nodes Colored by Load Centrality')

for i in range(2,3):
    for j in range(1,3):
        axs[i, j].set_visible(False)

plt.tight_layout()
plt.show()
```



On these plots, the nodes are **sized** based on the value of the measures.

```

fig, axs = plt.subplots(3, 3, figsize=(18, 15))

nx.draw(HPC, pos=nx.spring_layout(HPC), ax=axs[0, 0], node_color=griff_colors[1], alpha=0.5, node_size=[v * 1000 for v in degree_cent.values()],
axs[0, 0].set_title('Network of Sister Cities - Nodes Size by Degree Centrality')

nx.draw(HPC, pos=nx.spring_layout(HPC), ax=axs[0, 1], node_color=slyth_colors[1], alpha=0.5, node_size=[v * 1000 for v in betweenness_cent.values()],
axs[0, 1].set_title('Network of Sister Cities - Nodes Size by Betweenness Centrality')

nx.draw(HPC, pos=nx.spring_layout(HPC), ax=axs[0, 2], node_color=griff_colors[1], alpha=0.5, node_size=[v * 1000 for v in closeness_cent.values()],
axs[0, 2].set_title('Network of Sister Cities - Nodes Size by Closeness Centrality')

nx.draw(HPC, pos=nx.spring_layout(HPC), ax=axs[1, 0], node_color=slyth_colors[1], alpha=0.5, node_size=[v * 1000 for v in eigenvector_cent.values()],
axs[1, 0].set_title('Network of Sister Cities - Nodes Size by Eigenvector Centrality')

nx.draw(HPC, pos=nx.spring_layout(HPC), ax=axs[1, 1], node_color=griff_colors[1], alpha=0.5, node_size=[v * 1000 for v in pagerank_cent.values()],
axs[1, 1].set_title('Network of Sister Cities - Nodes Size by Pagerank Centrality')

nx.draw(HPC, pos=nx.spring_layout(HPC), ax=axs[1, 2], node_color=slyth_colors[1], alpha=0.5, node_size=[v for v in harmonic_cent.values()],
axs[1, 2].set_title('Network of Sister Cities - Nodes Size by Harmonic Centrality \n (These values are much larger than for the other metrics)')

nx.draw(HPC, pos=nx.spring_layout(HPC), ax=axs[2, 0], node_color=griff_colors[1], alpha=0.5, node_size=[v * 1000 for v in load_cent.values()],
axs[2, 0].set_title('Network of Sister Cities - Nodes Size by Load Centrality')

for i in range(2,3):
    for j in range(1,3):
        axs[i, j].set_visible(False)

plt.tight_layout()
plt.show()

```

5. Network Visualizations

The goal is to visualize the whole network HPC based on different centrality measures. In some cases, it is necessary to change the ratios, so that the nodes are more easily visible.

```
!pip install pyvis
from pyvis.network import Network
```

```
Collecting pyvis
  Downloading pyvis-0.3.2-py3-none-any.whl (756 kB)
    756.0/756.0 kB 7.1 MB/s eta 0:00:00
Requirement already satisfied: ipython>=5.3.0 in /usr/local/lib/python3.10/dist-packages (from pyvis) (7.34.0)
Requirement already satisfied: Jinja2>=2.9.6 in /usr/local/lib/python3.10/dist-packages (from pyvis) (3.1.3)
Requirement already satisfied: jsonpickle>=1.4.1 in /usr/local/lib/python3.10/dist-packages (from pyvis) (3.0.4)
Requirement already satisfied: networkx>=1.11 in /usr/local/lib/python3.10/dist-packages (from pyvis) (3.3)
Requirement already satisfied: setuptools>=18.5 in /usr/local/lib/python3.10/dist-packages (from ipython>=5.3.0->pyvis) (67.7.2)
Collecting jedi>=0.16 (from ipython>=5.3.0->pyvis)
  Downloading jedi-0.19.1-py2.py3-none-any.whl (1.6 MB)
    1.6/1.6 MB 11.3 MB/s eta 0:00:00
Requirement already satisfied: decorator in /usr/local/lib/python3.10/dist-packages (from ipython>=5.3.0->pyvis) (4.4.2)
Requirement already satisfied: pickleshare in /usr/local/lib/python3.10/dist-packages (from ipython>=5.3.0->pyvis) (0.7.5)
Requirement already satisfied: traitlets>=4.2 in /usr/local/lib/python3.10/dist-packages (from ipython>=5.3.0->pyvis) (5.7.1)
Requirement already satisfied: prompt-toolkit!=3.0.0,!<3.0.1,<3.1.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from ipython>=5.3.0->pyvis) (2.16.1)
Requirement already satisfied: pygments in /usr/local/lib/python3.10/dist-packages (from ipython>=5.3.0->pyvis) (2.16.1)
Requirement already satisfied: backcall in /usr/local/lib/python3.10/dist-packages (from ipython>=5.3.0->pyvis) (0.2.0)
Requirement already satisfied: matplotlib-inline in /usr/local/lib/python3.10/dist-packages (from ipython>=5.3.0->pyvis) (0.1.7)
Requirement already satisfied: pexpect>4.3 in /usr/local/lib/python3.10/dist-packages (from ipython>=5.3.0->pyvis) (4.9.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from Jinja2>=2.9.6->pyvis) (2.1.5)
Requirement already satisfied: parso<0.9.0,>=0.8.3 in /usr/local/lib/python3.10/dist-packages (from jedi>=0.16->ipython>=5.3.0->pyvis)
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.10/dist-packages (from pexpect>4.3->ipython>=5.3.0->pyvis)
Requirement already satisfied: wcwidth in /usr/local/lib/python3.10/dist-packages (from prompt-toolkit!=3.0.0,!<3.0.1,<3.1.0,>=2.0.0->ipython>=5.3.0->pyvis)
Installing collected packages: jedi, pyvis
Successfully installed jedi-0.19.1 pyvis-0.3.2
```

```
net = Network(notebook=True, width="100vw", height="100vh", bgcolor='#000000', font_color="white", cdn_resources='in_line')
nx.set_node_attributes(HPC, {node: degree * 100 for node, degree in degree_cent.items()}, "size")
nx.set_node_attributes(HPC, slyth_colors[1], "color")
net.from_nx(HPC)
net.show("harry_potter_degree.html")
```

```
harry_potter_degree.html
```



```
net = Network(notebook=True, width="100vw", height="100vh", bgcolor='#000000', font_color="white", cdn_resources='in_line')
nx.set_node_attributes(HPC, {node: degree * 100 for node, degree in betweenness_cent.items()}, "size")
nx.set_node_attributes(HPC, griff_colors[1], "color")
net.from_nx(HPC)
net.show("harry_potter_betweenness.html")
```

```
harry_potter_betweenness.html
```