


# Shakespearean Character Network Dataset

Kim, Heejin\*

 0000-0003-4522-8968

---

## Table of Contents

1. Data
  - 1.1 Dataset Description
  - 1.2 Dataset Structure and Content
2. Running the Script
3. Reuse Potential

## Abstract

The “Shakespearean Character Network” dataset leverages XML editions of Shakespeare’s plays from the Folger Shakespeare Library to analyze character interactions and dynamics within the plays. These XML files, containing detailed textual data such as dialogue and stage directions, are processed using the Python script in the repository. The script generates matrices that document character presence on stage and their verbal exchanges, stored in various directories such as `output_onstage` and `output_exchange`. Additionally, visualizations like heatmaps and network graphs offer visual and quantifiable insights into character co-presence and communication patterns. Centrality measures and clustering indices, computed for these interaction networks, further enhance the analysis by quantifying the degree of character clustering and the intensity of their interactions. The dataset aims to provide a comprehensive view of the structural relationships in Shakespeare’s plays. This resource is for researchers aiming to explore the dynamics of Shakespearean characters through a combination of computational methods and literary analysis.

**Keywords:** Character Network Analysis, Shakespeare, Digital Humanities, Dramatic Structure.

## 1. Data

### 1.1 Dataset Description

**Repository Location:** Github

**Repository Name:** `shakespearean_character_network`

**Dataset Location:**

[https://github.com/hkim1596/shakespearean\\_character\\_network](https://github.com/hkim1596/shakespearean_character_network)

[https://hkim1596.github.io/shakespearean\\_character\\_network](https://hkim1596.github.io/shakespearean_character_network)

**Creation Date:** May 15, 2024

---

\*Author: Department of English Language and Literature, Kyungpook National University, 80 Daehak-ro, Buk-gu, Daegu, Republic of Korea, [hkim1596@gmail.com](mailto:hkim1596@gmail.com)

**Dataset Creator:** Heejin Kim

## 1.2 Dataset Structure and Content

### **data Directory**

The dataset is based on Folger Shakespeare editions encoded as XML documents, which are freely available on their website: <https://www.folger.edu/explore/shakespeares-works/download>.<sup>1</sup> These XML editions are stored in the data directory of the GitHub repository. The XML documents represent digital transcriptions of Shakespeare's plays and are used to encode documents in a format that is both human-readable and machine-readable. The Folger Shakespeare editions in XML format provide detailed textual data, including character dialogue, stage directions, and other critical annotations. Importantly, these XML editions offer accurate and machine-readable information on the entrance and exit of characters, which is crucial for analyzing character interactions and dynamics. These files are essential for the dataset as they offer a standardized, rich textual source that can be parsed and analyzed to extract character interactions and other relevant information for the project. By utilizing these XML files, the dataset ensures accuracy and consistency in representing Shakespeare's original texts.

### **output\_onstage Directory**

The `output_onstage` directory contains files that document the onstage interactions of characters across Shakespeare's plays. These files in `.csv` (Comma-Separated Values) format are created using the `main.py` script, which processes XML files from the Folger Shakespeare Library to extract and quantify character interactions. The script analyzes each line to identify which characters are present together on stage and then constructs matrices representing these interactions. In these matrices, the rows correspond to characters and the columns represent Folger Through Line Numbers (FTLN). The FTLN columns indicate specific points in the narrative, effectively mapping the temporal sequence of the play. The cell values within these matrices denote the presence or absence of characters at particular narrative points, with 0 representing absence and 1 representing presence.

### **output\_onstage\_heatmap Directory**

The `output_onstage_heatmap` directory contains heatmap visualizations that depict the onstage co-presence of characters. These heatmaps are generated from the data in the `output_onstage` matrices, providing a visual representation of how often characters appear together in scenes. The heatmaps facilitate a quick and intuitive understanding of character interactions and dynamics, making it easier to identify clusters of characters based on their onstage or offstage status. HTML files in the directory can be opened using any standard web browser, allowing for an interactive exploration of the visualizations. Additionally, the `.png` files in the directory are non-interactive versions of the `.html` files with fewer details. These static images can be easily viewed without the need for a web browser.

### **output\_onstage\_matrix Directory**

The `output_onstage_matrix` directory contains `.csv` files that provide interaction matrices derived from the matrices in the `output_onstage` directory. Each file contains a matrix where both the rows and columns correspond to the characters in a play. The cells in these matrices quantify the number of lines in which each pair of characters appears together, allowing for a further analysis of character co-presence and interactions throughout the play. These matrices are essential for conducting network analysis and understanding the structural relationships between characters in Shakespeare's plays.

Additionally, the directory includes `.html` files, along with `.png` files with fewer details, that visualize these interaction matrices using the NetworkX library. NetworkX is a Python library used for the creation, manipulation,

and study of complex networks of nodes and edges.<sup>2</sup> The visualizations employ a spring layout, which simulates a physical system where nodes repel each other while edges act as springs pulling related nodes together.<sup>3</sup> This layout helps in creating aesthetically pleasing and informative representations of the network structure. In the visualizations, the shape of the nodes represents different groups of characters identified using the K-means clustering method with  $k = 2$ , while the color of the nodes indicates different groups of characters determined by the Louvain method.<sup>4</sup> The K-means method partitions the characters into two clusters based on their interactions, while the Louvain method, a community detection algorithm, identifies clusters of characters that are more densely connected internally than with the rest of the network. This combination of methods provides a comprehensive view of character dynamics and their social structure within the plays.

#### **output\_onstage\_matrix\_modified Directory**

The `output_onstage_matrix_modified` directory contains modified versions of the interaction matrices found in the `output_onstage_matrix` directory. These modified matrices have been refined to remove characters who are mostly isolated with almost no interactions with other characters, such as prologue, epilogue, and citizens 1, 2, and 3 in *Richard III*. These outliers can interfere with community detection methods by appearing as standalone communities, making it difficult to understand the clustering patterns of the majority of characters. By removing these outliers, the matrices in this directory provide a clearer and more useful representation of the main character interactions, thereby facilitating more effective network analysis and community detection.

#### **output\_exchange Directory**

The `output_exchange` directory contains matrices that document the amount of dialogue exchanged between characters. Unlike the `output_onstage_matrix` files, which focus on the presence of characters on stage together, the `output_exchange` matrices quantify the number of words spoken between each pair of characters. The `main.py` script processes the XML files in the data directory to calculate these exchanges. It assigns speakers and listeners for each line of dialogue by parsing the text and identifying who is speaking. The assumption made is that all characters on stage are listeners while a speaker gives his or her lines. In these matrices, the columns represent the speakers, while the rows represent the listeners. The cell values indicate the total number of words spoken by one character to another, offering a detailed view of verbal interactions. These matrices are essential for understanding the communication dynamics within the plays, highlighting which characters interact the most through dialogue. By analyzing these dialogue exchanges, researchers can gain insights into the relationships and influence of characters based on their verbal interactions. The data in the `output_exchange` matrices provide a complementary perspective to the onstage presence data, offering a more nuanced view of character interactions within Shakespeare's plays.

#### **output\_exchange\_heatmap Directory**

The `output_exchange_heatmap` directory contains heatmap visualizations of the dialogue exchanges between characters. These heatmaps are generated from the data in the `output_exchange` matrices and provide a visual representation of the intensity and frequency of verbal interactions. The heatmaps help to quickly identify which characters engage most frequently in dialogue, making it easier to discern communication patterns and key relationships within the plays.

#### **output\_exchange\_modified Directory**

The `output_exchange_modified` directory contains modified versions of the interaction matrices found in the `output_exchange` directory. These files have been refined similarly to those in the `output_onstage_matrix_modified` directory, removing characters who are mostly isolated and have minimal interactions with other characters.

### **output\_scores Directory**

The `output_scores` directory contains files that calculate various network centrality measures and clustering indices for both exchange and onstage interaction matrices. Specifically, the degree centralization scores, betweenness centrality scores, closeness centrality scores, eigenvector centrality scores, Dunn Index, and silhouette scores are computed for the networks derived from the character interactions. Degree centralization measures how centralized the network is by evaluating the variance of node degrees, with the formula summing the differences between the maximum degree and each node's degree, normalized by the theoretical maximum. Betweenness centralization identifies nodes that act as bridges by measuring the number of shortest paths passing through each node, summing the differences between the maximum betweenness centrality and each node's betweenness, and normalizing this sum. Closeness centralization assesses how quickly information spreads from a given node to others by calculating the average shortest path length. The centralization score sums the differences between the maximum closeness centrality and each node's closeness, then normalizes the total. Eigenvector centralization evaluates the influence of nodes based on their connections to other high-scoring nodes, summing the differences between the maximum eigenvector centrality and each node's eigenvector centrality, and normalizing the total. The Dunn Index and silhouette scores are used to evaluate the quality of clusters formed by the character interactions. The Dunn Index is calculated as the ratio of the minimum inter-cluster distance to the maximum intra-cluster distance, highlighting the compactness and separation of clusters.

These scores are visualized using bar graphs and box plots, generated by the `plot_scores` function in the `main.py` script. In the bar graphs, the x-axis represents the different plays, ordered chronologically according to the conjectured year of composition as per the *New Oxford Shakespeare* edition to maintain consistency and ease of comparison across the visualizations.<sup>5</sup> The y-axis shows the respective centrality or clustering score, allowing for a comparative view of how different plays rank in terms of their network metrics. Different colors are used for the bars to distinguish between genres of the plays: comedies are colored light blue, tragedies are colored slate gray, histories are colored red, and plays with an unidentified genre are colored light gray. This color-coding facilitates an immediate visual distinction between different genres, enhancing the interpretability of the graphs. The box plots provide additional insights into the distribution and variability of these scores across all plays. For each centrality measure and clustering index, the box plots depict the median, quartiles, and potential outliers, offering a more detailed statistical summary. These visualizations facilitate a comprehensive understanding of the structural and communicative dynamics in Shakespeare's plays, highlighting variations in character interactions and network properties across different works.

### **Python Code: main.py**

The `main.py` script in the repository (see fig. 1) serves as the primary engine for processing and analyzing character interactions within Shakespeare's plays. It begins by importing necessary libraries such as `scikit-learn`, `Community API`, and `NetworkX`, and defining key functions for parsing XML files from the Folger Shakespeare Library. The script reads these XML files to extract detailed information about character dialogues and stage directions. By parsing this data, `main.py` generates various data structures, including onstage presence matrices and interaction matrices, which are then stored in the appropriate directories. These matrices quantify character co-presence and dialogue exchanges, providing a foundation for further analysis. The script also includes functions to compute network centrality measures and clustering indices, allowing for a detailed exploration of the character networks within the plays.

```

51 # Iterate through all nodes in the XML file
52 for tag in soup.find_all(['sp', 'stage', 'milestone']):
53     # Update current time from milestone using xml:id attribute
54     if tag.name == 'milestone' and tag.get('unit') == 'ftln':
55         current_time = tag['xml:id'] # Update current time
56         # Record characters on stage for the current time
57         stage_timeline[current_time] = on_stage.copy() # Use copy to avoid reference issues
58
59     # Stage direction for entrance
60     if tag.name == 'stage' and tag.get('type') == 'entrance' and 'who' in tag.attrs:
61         characters_entering = [re.sub(r"_[^\s,]*", "", char.replace("#", "")) for char in tag['who'].
62             split()]
63         on_stage = list(set(on_stage + characters_entering))
64
65     # Stage direction for exit
66     if tag.name == 'stage' and tag.get('type') == 'exit' and 'who' in tag.attrs:
67         characters_exiting = [re.sub(r"_[^\s,]*", "", char.replace("#", "")) for char in tag['who'].
68             split()]
69         on_stage = [char for char in on_stage if char not in characters_exiting]
70
71     # Speech tag
72     if tag.name == 'sp' and 'who' in tag.attrs: # Check if the tag is a speech
73         speakers = [re.sub(r"_[^\s,]*", "", speaker.replace("#", "")) for speaker in tag['who'].split()]
74         for speaker in speakers: # Loop through each speaker
75             others = [char for char in on_stage if char != speaker] # Listeners for this specific
76                 speaker
77             if not others: # If there are no listeners, skip to the next speaker
78                 others.append(speaker) # Add the speaker to the list of listeners
79
80             word_count = count_words(tag) // len(speakers) # Divide the word count among the speakers
81
82             for listener in others: # Loop through each listener
83                 character_exchange.setdefault(speaker, {}).setdefault(listener, 0) # Add speaker and
84                 listener to the dictionary if they don't exist
85                 character_exchange[speaker][listener] += word_count # Add the word count to the
86                 speaker-listener pair
87
88     # Convert interactions dictionary to dataframe and transpose
89     character_exchange_df = pd.DataFrame(character_exchange).T.fillna(0) # Convert the dictionary to a
90     dataframe and transpose it
91
92     all_characters = list(set(character_exchange_df.index).union(set(character_exchange_df.columns))) # Get
93     all characters
94     character_exchange_df = character_exchange_df.reindex(index=all_characters, columns=all_characters).fillna
95     (0) # Reindex the dataframe and fill missing values with 0

```

Figure 1. Snippet of code in main.py

## 2. Running the Script

To run main.py, users need to ensure that Python and the required libraries are installed, then execute the script from the command line with `python main.py`. This process will generate the corresponding output files in the specified directories. Researchers can modify the script by adjusting the input data, altering the parsing logic, or customizing the analysis and visualization methods to suit specific research needs. This flexibility enables a tailored exploration of various aspects of Shakespeare’s plays, enhancing the understanding of character interactions and dynamics.

## 3. Reuse Potential

The “Shakespearean Character Network” dataset is openly available for anyone to use, providing a resource for writing papers and conducting various forms of research. For instance, my own article titled “Shakespearean Character Network Analysis: Residual Dramatic Structures of Medieval Touring Companies and Neoclassical Unity” utilizes a part of the dataset to investigate structural unity in Renaissance drama.<sup>6</sup> The paper employs

network analysis to trace the influence of medieval touring companies on 16th-century dramatic structures, using community detection algorithms and silhouette scores to analyze bipartite or multipartite structures across 38 Shakespearean plays. This aims to demonstrate the dataset's versatility and potential for exploring diverse research questions within the fields of digital humanities and literary analysis. Researchers are encouraged to leverage this dataset to further their understanding of character interactions, narrative structures, and other aspects of Shakespeare's plays. The flexibility and depth of the dataset make it a tool for enhancing the scope and depth of digital humanities research.

---

<sup>1</sup> The Folger Shakespeare digital edition is widely used in Shakespeare studies and digital humanities projects. See Folgert Karsdorp, Mike Kestemont, and Allen Riddell. (2021). *Humanities Data Analysis - Case Studies with Python*. Princeton University Press.

<sup>2</sup> Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. (2008). "Exploring Network Structure, Dynamics, and Function Using NetworkX." *Proceedings of the 7th Python in Science Conference* 11–15.

<https://networkx.org/documentation/stable/reference>

<sup>3</sup> [https://networkx.org/documentation/stable/reference/generated/networkx.drawing.layout.spring\\_layout.html](https://networkx.org/documentation/stable/reference/generated/networkx.drawing.layout.spring_layout.html)

<sup>4</sup> <https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html>

<https://python-louvain.readthedocs.io/en/latest/api.html>

<sup>5</sup> Gary Taylor and Gabriel Egan, eds. (2017). *The New Oxford Shakespeare Authorship Companion*. Oxford University Press.

<sup>6</sup> Heejin. Kim (2024). "Shakespearean Character Network Analysis: Residual Dramatic Structures of Medieval Touring Companies and Neoclassical Unity." *In/Outside* 56 18-56. <https://doi.org/10.46645/inoutsesk.56.1>

## References

Hagberg, Aric A., Daniel A. Schult, and Pieter J. Swart. (2008). "Exploring Network Structure, Dynamics, and Function Using NetworkX." *Proceedings of the 7th Python in Science Conference* 11–15.

Karsdorp, Folgert, Mike Kestemont, and Allen Riddell. (2021). *Humanities Data Analysis - Case Studies with Python*. Princeton University Press.

Kim, Heejin. (2024). "Shakespearean Character Network Analysis: Residual Dramatic Structures of Medieval Touring Companies and Neoclassical Unity." *In/Outside* 56 18-56.

Taylor, Gary, and Gabriel Egan, eds. (2017). *The New Oxford Shakespeare Authorship Companion*. Oxford University Press.