Automated Information Extraction model, Character and Network Analysis in Texts

Data mining course final project
Dr. Hubert Chan

Mohsen Rahimi
Aug 2023

Introduction

In recent years, the growing availability of digital books and the advancement of computational methods have opened up new possibilities for analyzing unstructured text data, such as literary works, using natural language processing (NLP) and information extraction (IE) techniques, particularly in the field of character and network analysis.

This project aims to develop an automated model for analyzing the characters and their relationships in books using statistical and data mining techniques. The purpose of this model is to provide insights into the complex narrative structures and character dynamics found within literary works.

To achieve this goal, the project involves several stages, including data acquisition, preprocessing, named entity recognition (NER), evaluation of NER, co-occurrence matrix and graph creation, centrality measurements, network analysis, community detection, visualization, validation, interpretation, refinement, and automation. By leveraging various tools and methodologies, such as Python programming environment and machine learning algorithms, the model will be able to process and analyze large volumes of text data, identify characters and their relationships, and visualize the resulting networks.

For this project, the dataset selected is Jane Austen's classic novel, "Pride and Prejudice," which is downloaded from Gutenberg project. This novel serves as an excellent choice for developing an automated model for character relationship analysis. The novel features a diverse range of characters with complex and dynamic relationships, offering ample opportunities for analysis and exploration.

The model's performance will be assessed using appropriate evaluation metrics, such as precision,

recall, and F1-score. These metrics will help ensure the model's reliability and validity when analyzing character relationships and network structures within the text.
Ultimately, this project aims to contribute to the growing field of digital humanities by providing a novel, automated approach to character and network analysis in literature. By developing and applying this model to "Pride and Prejudice," the project seeks to enhance our understanding of narrative structures and character dynamics. This, in turn, enriches the reading experience and offers valuable insights for scholars, literary enthusiasts, and readers alike.

Technical methods, Results and Conclusions

1- Download and load the text.

"Pride and Prejudice" is an excellent choice for a character relationship analysis project. The novel features a diverse range of characters, each with their unique motivations, desires, and personalities. The relationships between these characters are complex and dynamic, providing ample opportunity for analysis.
Throughout the story, the characters' relationships evolve, and their interactions reveal much about their personalities and values. The novel's primary focus on themes like love, marriage, social class, and the role of women in society ensures that the characters' relationships are rich in substance and meaning.
By analyzing the relationships in "Pride and Prejudice," it will be explored how characters influence one another, how their social status impacts their interactions, and how their personal growth affects their relationships. Overall, it's a great choice for a character relationship and network analysis project.
To obtain the text, the novel is downloaded from the Project Gutenberg, a comprehensive repository of over 60,000 free eBooks, including many literary classics. The following code snippet demonstrates the process of downloading the book (with ID 1342) and saving it as a text file:

```
import requests    # download the book
import re
from transformers import AutoTokenizer, AutoModelForTokenClassification
from transformers import pipeline
import contractions
from itertools import combinations
from nltk.tokenize import sent_tokenize
from nltk.corpus import stopwords
import networkx as nx
from time import time
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/mohsenrahimi/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
True
```

Figure 1

```python
import requests    # download the book
book_id = 1342
url = f"https://www.gutenberg.org/files/{book_id}/{book_id}-0.txt"
response = requests.get(url)

# Check if the request was successful
if response.status_code == 200:
    with open("book.txt", "w", encoding="utf-8") as f:
        f.write(response.text)
    print("Book downloaded successfully.")
else:
    print("Failed to download the book.")
```

```
Book downloaded successfully.
```

```python
with open("book.txt", "r", encoding="utf-8") as f:
    book_content = f.read()                        #`book_content` now contains the entire text of the book
```

Figure 2

This code snippet in above first sends a request to the Project Gutenberg website to fetch the content of "Pride and Prejudice." After ensuring that the request is successful, it writes the text content into a local file named "book.txt." Finally, the content of the file is read into a variable named `book_content`, which now contains the entire text of the book, ready for further analysis and processing.

2- Preprocessing:

Text cleaning is an essential preprocessing step when working with raw text data, especially when dealing with a book. The goal is to remove any irrelevant information, inconsistencies, and noise from the text, making it easier to analyze.

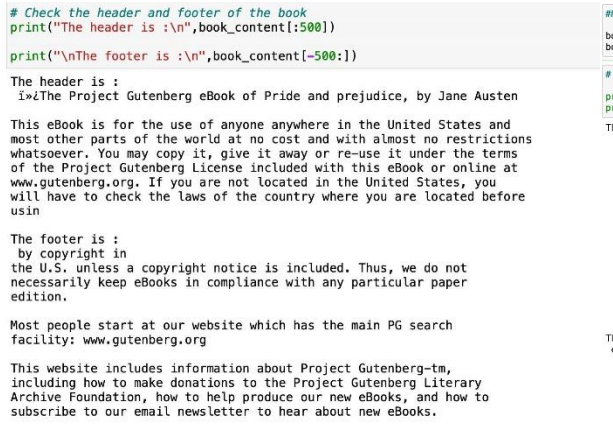
For character relationship analysis, we need to focus on cleaning the text by removing any irrelevant information such as page numbers, headers, footers, and any special characters that could introduce noise and inconsistencies in the text. This will help us focus on the actual content of the book.

2-1: Remove header and footer:

In the codes in below, several steps are taken to remove the header and footer of the text of "Pride and Prejudice":

1. Identify the header and footer of the book using the `print` function to visualize the the first and last 500 characters of the content.
2. Attempt to remove page numbers, headers, and footers using regular expressions. However, this approach does not yield the desired results.
3. As an alternative, manually identify the start and end phrases of the actual book content by printing specific character ranges.
4. Determine the starting phrase as "Chapter I.]" and the ending phrase as "THE END."
5. Extract the actual content of the book by slicing the text based on the identified start and end indices.

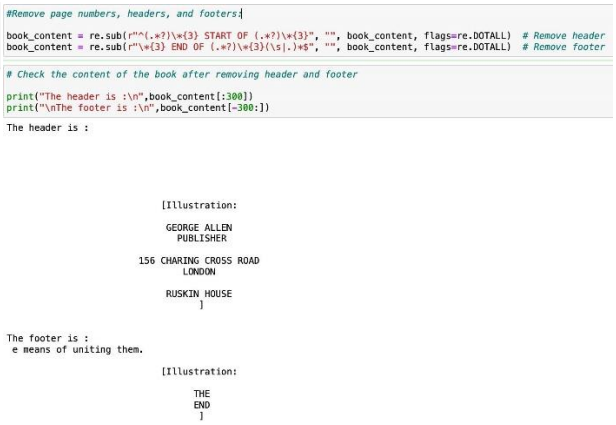
After these steps, the resulting `book_content` variable contains the cleaned text, free from headers, footers, and other irrelevant information. This cleaned text is now ready for further processing and analysis in the character relationship analysis project.



```
# Check the header and footer of the book
print("The header is :\n",book_content[:500])

print("\nThe footer is :\n",book_content[-500:])
```

```
The header is :
 ï»¿The Project Gutenberg eBook of Pride and prejudice, by Jane Austen

This eBook is for the use of anyone anywhere in the United States and
most other parts of the world at no cost and with almost no restrictions
whatsoever. You may copy it, give it away or re-use it under the terms
of the Project Gutenberg License included with this eBook or online at
www.gutenberg.org. If you are not located in the United States, you
will have to check the laws of the country where you are located before
usin

The footer is :
 by copyright in
the U.S. unless a copyright notice is included. Thus, we do not
necessarily keep eBooks in compliance with any particular paper
edition.

Most people start at our website which has the main PG search
facility: www.gutenberg.org

This website includes information about Project Gutenberg-tm,
including how to make donations to the Project Gutenberg Literary
Archive Foundation, how to help produce our new eBooks, and how to
subscribe to our email newsletter to hear about new eBooks.
```

Figure 3



```
#Remove page numbers, headers, and footers:

book_content = re.sub(r"^(.*?)\*{3} START OF (.*?)\*{3}", "", book_content, flags=re.DOTALL)  # Remove header
book_content = re.sub(r"\*{3} END OF (.*?)\*{3}(\s|.)*$", "", book_content, flags=re.DOTALL)  # Remove footer
```

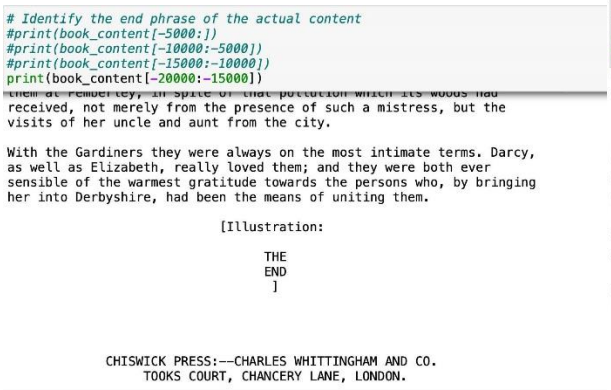
```
# Check the content of the book after removing header and footer

print("The header is :\n",book_content[:300])
print("\nThe footer is :\n",book_content[-300:])
```

```
The header is :




                                [Illustration:

                                 GEORGE ALLEN
                                   PUBLISHER

                               156 CHARING CROSS ROAD
                                     LONDON

                                  RUSKIN HOUSE
                                       ]


The footer is :
 e means of uniting them.

                                [Illustration:

                                     THE
                                     END
                                      ]
```

Figure 4

```
# Identify the start phrase
with open("book.txt", "r", encoding="utf-8") as f:
    book_content = f.read()                       #`book_content` now contains the entire text of the book

# In below in each time I printed 10,000 characters to find out the starting point

#print(book_content[1:30000])
print(book_content[30000:40000])
#print(book_content[40000:50000])
```

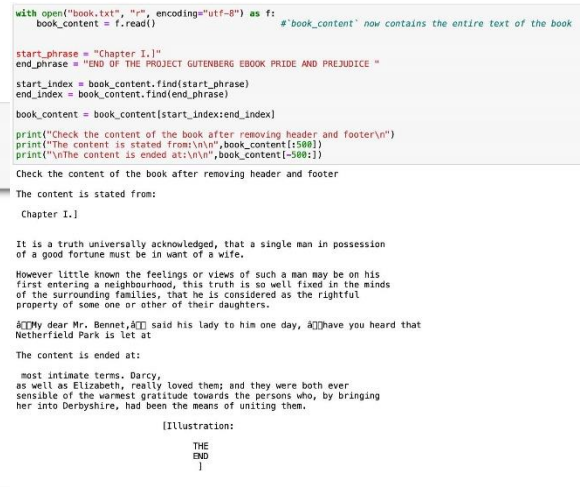
Figure 5



```
# Identify the end phrase of the actual content
#print(book_content[-5000:])
#print(book_content[-10000:-5000])
#print(book_content[-15000:-10000])
print(book_content[-20000:-15000])
```

```
them at Pemberley, in spite of that pollution which its woods had
received, not merely from the presence of such a mistress, but the
visits of her uncle and aunt from the city.

With the Gardiners they were always on the most intimate terms. Darcy,
as well as Elizabeth, really loved them; and they were both ever
sensible of the warmest gratitude towards the persons who, by bringing
her into Derbyshire, had been the means of uniting them.

                                [Illustration:

                                     THE
                                     END
                                      ]



                CHISWICK PRESS:--CHARLES WHITTINGHAM AND CO.
                     TOOKS COURT, CHANCERY LANE, LONDON.
```

Figure 6



```
with open("book.txt", "r", encoding="utf-8") as f:
    book_content = f.read()                       #`book_content` now contains the entire text of the book


start_phrase = "Chapter I.]"
end_phrase = "END OF THE PROJECT GUTENBERG EBOOK PRIDE AND PREJUDICE "

start_index = book_content.find(start_phrase)
end_index = book_content.find(end_phrase)

book_content = book_content[start_index:end_index]

print("Check the content of the book after removing header and footer\n")
print("The content is stated from:\n\n",book_content[:500])
print("\nThe content is ended at:\n\n",book_content[-500:])
```

```
Check the content of the book after removing header and footer

The content is stated from:

 Chapter I.]


It is a truth universally acknowledged, that a single man in possession
of a good fortune must be in want of a wife.

However little known the feelings or views of such a man may be on his
first entering a neighbourhood, this truth is so well fixed in the minds
of the surrounding families, that he is considered as the rightful
property of some one or other of their daughters.

â€œMy dear Mr. Bennet,â€ said his lady to him one day, â€have you heard that
Netherfield Park is let at

The content is ended at:

 most intimate terms. Darcy,
as well as Elizabeth, really loved them; and they were both ever
sensible of the warmest gratitude towards the persons who, by bringing
her into Derbyshire, had been the means of uniting them.

                                [Illustration:

                                     THE
                                     END
                                      ]
```

Figure 7

2-2: splitting it into chapters

In this stage of the preprocessing, the text is further refined by splitting it into chapters. This process is crucial for the character relationship analysis, as it allows for a more structured and

granular examination of character interactions within each chapter. By dividing the book into chapters, we can better identify the context of relationships and how they evolve throughout the narrative. The following steps are taken to split the text into chapters:
1. Add an extra "CHAPTER" keyword at the end of the book. This step is necessary to ensure that the last chapter is included correctly when splitting the text. By appending the "CHAPTER" keyword, a consistent splitting pattern is maintained for all chapters, including the last one.
2. Split the book into chapters using the "CHAPTER" keyword. The `split` function breaks the text into a list of chapters based on the occurrences of the "CHAPTER" keyword.

After these stages, the text is organized into chapters, facilitating a more detailed and context-aware analysis of character relationships and their development throughout the story.

```
book_content_with_last_chapter = book_content + "CHAPTER"

# Split the book into chapters using the "CHAPTER" keyword
chapters = book_content_with_last_chapter.split("CHAPTER")
# Remove the last element, which is just the text after the last chapter
chapters.pop()
# Print the number of chapters and a sample chapter
print(f"Total chapters: {len(chapters)}")
```

```
Total chapters: 60
```

Figure 8

2-3: removing any special characters:

In this stage of the preprocessing, the focus is on cleaning the content of each chapter by removing any special characters that might interfere with the character relationship analysis. By applying a cleaning function to all chapters, we ensure that the text is free from any unnecessary noise and can be effectively analyzed. The following steps are taken in this stage:
1. Define a `clean_chapter_content` function that takes a chapter's content as input and utilizes regular expressions to remove any characters that are not letters, spaces, newlines, or common punctuation marks (such as period, comma, semicolon, exclamation mark, question mark, single quote, and double quote).
2. Apply the cleaning function to all chapters by using a list comprehension. This step creates a new list, `chapters_clean`, containing the cleaned content of each chapter.
3. Print a sample cleaned chapter to confirm that the cleaning process has been successful and that the text is ready for further processing and analysis.

```python
import re
# Function to clean the content of a chapter
def clean_chapter_content(chapter_content):
    return re.sub(r"[^a-zA-Z\s\n.,;!?'\"]", "", chapter_content)

# Apply the cleaning function to all chapters
chapters_clean = [clean_chapter_content(chapter) for chapter in chapters]

# Print a sample cleaned chapter
print("Cleaned Chapter 1 as below:\n", chapters_clean[0][:300])
```

```
Cleaned Chapter 1 as below:
 Chapter I.


It is a truth universally acknowledged, that a single man in possession
of a good fortune must be in want of a wife.

However little known the feelings or views of such a man may be on his
first entering a neighbourhood, this truth is so well fixed in the minds
of the surrounding famili
```

Figure 9

2-4: Expanding contractions within the text:

In this stage focus is on expanding contractions within the text to ensure a more standardized and consistent representation of the language. By converting contractions to their expanded forms (e.g., changing "don't" to "do not"). This preprocessing step ensures that the text is better suited for a detailed and accurate examination of character relationships and interactions throughout the novel. The following steps are taken in this stage (Figure 6 and Figure 7):

1. Import the `contractions` library, which provides a convenient way to expand contractions within the text.
2. Apply the `contractions.fix()` function to each chapter using a list comprehension. This step creates a new list, `chapters_no_contractions`, containing the content of each chapter with contractions expanded.
3. Print a sample chapter without contractions to confirm that the expansion process has been successful.

```python
import contractions

chapters_no_contractions = [contractions.fix(chapter) for chapter in chapters_clean]

print("Chapter 1 without contractions:\n", chapters_no_contractions[0][:500])
```

```
Chapter 1 without contractions:
 Chapter I.


It is a truth universally acknowledged, that a single man in possession
of a good fortune must be in want of a wife.

However little known the feelings or views of such a man may be on his
first entering a neighbourhood, this truth is so well fixed in the minds
of the surrounding families, that he is considered as the rightful
property of some one or other of their daughters.

My dear Mr. Bennet, said his lady to him one day, have you heard that
Netherfield Park is let at last?
```

Figure 10

2-5: Concatenating the lines

In this stage of the preprocessing, the focus is on concatenating the lines within each chapter to create a continuous text representation. By converting the separate lines into a single, continuous string, we streamline the text for easier processing and analysis of character relationships and interactions. This preprocessing step ensures that the text is optimally structured for the subsequent stages of the character relationship analysis. The following steps are taken in this stage:

1. Use a list comprehension to iterate through each chapter in chapters_no_contractions. For each chapter, apply the splitlines() method to break the text into a list of lines, and then use the join() method with a space separator to concatenate the lines back into a single string. This step creates a new list, chapters_continuous, containing the continuous text representation of each chapter.

2. Print a sample chapter to confirm that the concatenation process has been successful.

```
# Concatenate the lines in each chapter
chapters_continuous = [' '.join(chapter.splitlines()) for chapter in chapters_no_contractions]

# Print a sample chapter
print("Chapter 1:\n", chapters_continuous[0][:500])
```

```
Chapter 1:
 Chapter I.   It is a truth universally acknowledged, that a single man in possession of a good fortune must be in w
ant of a wife.  However little known the feelings or views of such a man may be on his first entering a neighbourhoo
d, this truth is so well fixed in the minds of the surrounding families, that he is considered as the rightful prope
rty of some one or other of their daughters.  My dear Mr. Bennet, said his lady to him one day, have you heard that
Netherfield Park is let at last?  Mr.
```

Figure 11

2-6: Tokenization:

Now the focus is on tokenizing the continuous text of each chapter into sentences. By breaking the text into sentences, we can better analyze character relationships and interactions within the context of individual statements and exchanges. The following steps are taken in this stage:

1. Import the sent_tokenize function from the nltk.tokenize module, which is a powerful tool for breaking text into sentences.

2. Use a list comprehension to apply the sent_tokenize function to each chapter in the chapters_continuous list. This step creates a new list, chapters_sentences, containing the tokenized sentences of each chapter.

3. Print the first 10 tokenized sentences from Chapter I to confirm that the sentence tokenization process has been successful and that the text is ready for further processing and analysis.

```
from nltk.tokenize import sent_tokenize
# Apply sentence tokenization to all chapters
chapters_sentences = [sent_tokenize(chapter) for chapter in chapters_continuous]

# Printthe first 10 tokenized sentences from Chapter I
chapters_sentences[0][:10]
```

```
['Chapter I.',
 'It is a truth universally acknowledged, that a single man in possession of a good fortune must be in want of a wif
e.',
 'However little known the feelings or views of such a man may be on his first entering a neighbourhood, this truth
is so well fixed in the minds of the surrounding families, that he is considered as the rightful property of some on
e or other of their daughters.',
 'My dear Mr. Bennet, said his lady to him one day, have you heard that Netherfield Park is let at last?',
 'Mr. Bennet replied that he had not.',
 'But it is, returned she; for Mrs. Long has just been here, and she told me all about it.',
 'Mr. Bennet made no answer.',
 'Do not you want to know who has taken it?',
 'cried his wife, impatiently.',
 'You want to tell me, and I have no objection to hearing it.']
```

Figure 12

2-7: Avoid doing these techniques in preprocessing:

In character relationship analysis, preserving the original forms of words, context, grammatical structure, and dialogue is crucial. Therefore, we want to avoid or modify certain preprocessing steps, such as stemming and stop word removal, to maintain the integrity of the character interactions and relationships in the text.

• Convert text to lowercase:

Converting all text to lowercase can help standardize the text and make it easier to analyze. However, as we are analyzing character relationships, need to preserve the capitalization of character names to distinguish them from common words.

• Removing stop words:

Stop words are common words such as 'the', 'and', 'is', etc., that do not carry much meaning and are often removed to reduce noise and improve the efficiency of text analysis. However, in character relationship analysis, removing stop words might not be necessary because dialogue and context are essential for understanding character relationships. Removing stop words might lead to loss of valuable information and context.

• Stemming and lemmatization:

These techniques involve reducing words to their root forms by removing inflections and derivational affixes. While stemming and lemmatization can be useful in tasks like text classification or topic modeling, they might not be relevant for character relationship analysis. Preserving the original forms of words is essential for accurately identifying character names and understanding dialogue between characters.

3- Named Entity Recognition (NER) pipeline

In this stage, we create a Named Entity Recognition (NER) pipeline using the Hugging Face Transformers library. This pipeline will be used to process the text and identify character names in "Pride and Prejudice." By leveraging a pre-trained BERT-based NER model, we can efficiently and accurately recognize characters and their interactions within the novel.

Why choose a pre-trained BERT??!!

1. BERT is a highly effective and powerful language model that has shown state-of-the-art performance in various NLP tasks, including NER.

2. Using a pre-trained model saves time and computational resources, as it has already learned general language patterns and features from vast amounts of data.

3. Fine-tuning a pre-trained BERT model for the specific NER task allows for high accuracy and adaptability, ensuring reliable character identification in the text.

The following steps are taken in this stage:

1. Load a pre-trained BERT-based NER model by specifying the model name and using the AutoTokenizer.from_pretrained() and AutoModelForTokenClassification.from_pretrained() functions.

2. Create a NER pipeline using the pipeline() function from the Transformers library. This pipeline will utilize the loaded model and tokenizer for processing text.

3. Define a helper function merge_subwords() to handle cases where character names are split into subwords. This function takes a list of entities and merges any subwords that start with "##" into the previous word.

4. Define another helper function is_valid_entity() to filter out valid character entities from the list of identified entities. This function checks if the entity is of type "I-PER" (person), not a common title, not a stop word, and has a length greater than 2 characters.

By completing this stage, we have set up a powerful NER pipeline capable of identifying character names in "Pride and Prejudice." This pipeline will be instrumental in analyzing character relationships and interactions throughout the text, enabling us a deeper understanding of the story's dynamics and character development.

```python
# Load a pre-trained BERT-based NER model
tokenizer = AutoTokenizer.from_pretrained("dbmdz/bert-large-cased-finetuned-conll03-english")
model = AutoModelForTokenClassification.from_pretrained("dbmdz/bert-large-cased-finetuned-conll03-english")

ner_pipeline = pipeline("ner", model=model, tokenizer=tokenizer)    # Create a NER pipeline


def merge_subwords(entities):
    merged_entities = []
    for entity in entities:
        if entity["word"].startswith("##"):
            if len(merged_entities) > 0:
                merged_entities[-1]["word"] += entity["word"][2:]
        else:
            merged_entities.append(entity)
    return merged_entities



def is_valid_entity(entity):
    common_titles = ["Mrs", "Mr", "Miss", "Dr", "Sir", "Pope", "Lady", "."]
    stop_words = set(stopwords.words('english'))

    return (
        entity["entity"] == "I-PER"
        and entity["word"] not in common_titles
        and entity["word"] not in stop_words
        and len(entity["word"]) > 2  # Exclude single alphabets
    )
```

Figure 13

4- Evaluation of NER

We can calculate the accuracy of the NER model by comparing the detected character names to a set of ground truth character names. For this, a list of correct character names from the text is needed, which is created manually. Stages are in the following:

1. Create a set of ground truth character names.

2. Calculate true positives (TP), false positives (FP), and false negatives (FN).

    - True Positives (TP): Named entities correctly identified by model

    - False Positives (FP): Named entities incorrectly identified by model

    - False Negatives (FN): Named entities missed by model

3. Calculate precision, recall, and F1-score.

    - Precision = TP / (TP + FP)

    - Recall = TP / (TP + FN)

    - F1 Score = 2 * (Precision * Recall) / (Precision + Recall)

And this is ground truth character names: ground_truth_names= ["Bennet", "Long", "George", "Morris", "Michaelmas", "Bingley", " Mrs. Bennet", "William", "Lucas", "Lizzy", "Jane", "Lydia", "Elizabeth", "Kitty", "Mary", "Hurst", "Darcy", "Catherine", "King", "Boulanger"]

And now in the code below the character names by NER model are extracted:

```
unique_entities = set()
for chapter_idx, sentences in enumerate(chapters_sentences[:3]):
    print(f"Processing Chapter {chapter_idx + 1}")
    for sentence in sentences:
        # Extract named entities from the sentence
        entities = ner_pipeline(sentence)

        # Merge subwords and clean up the named entities
        merged_entities = merge_subwords(entities)

        # Filter out unwanted entity types and common titles
        filtered_entities = [entity for entity in merged_entities if is_valid_entity(entity)]

        # Add the named entities to the set
        for entity in filtered_entities:
            unique_entities.add(entity["word"])

# Print the unique named entities
print("Unique Named Entities:")
for entity in unique_entities:
    print(entity)
```

```
Processing Chapter 1
Processing Chapter 2
Processing Chapter 3
Unique Named Entities:
Bennets
Allen
Catherine
Lucas
Morris
King
Bennet
Lydia
Darcy
Lizzy
Elizabeth
Jane
Hurst
George
Maria
Long
Mary
William
Kitty
Bingley
```

Figure 14

detected_NER= ['Bennets', 'Allen', 'Catherine', 'Lucas', 'Morris', 'King', 'Bennet', 'Lydia', 'Darcy', 'Lizzy', 'Elizabeth', 'Jane', 'Hurst', 'George', 'Maria', 'Long', 'Mary', 'William', 'Kitty', 'Bingley']

These code snippets in below define two functions, evaluate_ner_results() and calculate_metrics(), which are used to assess the performance of the NER pipeline.

1. evaluate_ner_results() takes detected NER results and ground truth character names as input, calculates true positives (TP), false positives (FP), and false negatives (FN), and returns a DataFrame containing the evaluation results.

2. calculate_metrics() takes the evaluation results as input and computes the precision, recall, and F1-score, returning a DataFrame containing these metrics.

3. The final part of the code applies these functions to the NER pipeline results, displaying the evaluation results and performance metrics for better understanding of the pipeline's accuracy and effectiveness.

```
def evaluate_ner_results(detected_ner, ground_truth_names):
    tp = set(detected_ner).intersection(ground_truth_names)
    fp = set(detected_ner).difference(ground_truth_names)
    fn = set(ground_truth_names).difference(detected_ner)

    evaluation_results = {
        "TP": len(tp),
        "FP": len(fp),
        "FN": len(fn),
        "true_positives": [list(tp)],
        "false_positives": [list(fp)],
        "false_negatives": [list(fn)],
    }

    return pd.DataFrame(evaluation_results)

def calculate_metrics(evaluation_results):
    precision = evaluation_results.at[0, "TP"] / (evaluation_results.at[0, "TP"] + evaluation_results.at[0, "FP"])
    recall = evaluation_results.at[0, "TP"] / (evaluation_results.at[0, "TP"] + evaluation_results.at[0, "FN"])
    f1_score = 2 * (precision * recall) / (precision + recall)

    metrics = {
        "precision": [precision],
        "recall": [recall],
        "f1_score": [f1_score],
    }

    return pd.DataFrame(metrics)

evaluation_results = evaluate_ner_results(detected_ner, ground_truth_names)
metrics = calculate_metrics(evaluation_results)
print("Evaluation Results of NER stage:")
display(evaluation_results)
print("Metrics:")
display(metrics)
```

```
Evaluation Results of NER stage:
```

	TP	FP	FN	true_positives	false_positives	false_negatives
0	17	3	3	[George, Long, Bennet, Mary, Catherine, Darcy,...	[Maria, Bennets, Allen]	[Michaelmas, Mrs.Bennet, Boulanger]

```
Metrics:
```

	precision	recall	f1_score
0	0.85	0.85	0.85

Figure 15

Evaluation Results of NER stage:

	TP	FP	FN	true_positives	false_positives	false_negatives
0	17	3	3	[George, Long, Bennet, Mary, Catherine, Darcy,...	[Maria, Bennets, Allen]	[Michaelmas, Mrs.Bennet, Boulanger]

Metrics:

	precision	recall	f1_score
0	0.85	0.85	0.85

Figure 16

The NER pipeline's evaluation results show a high recall (0.85) and precision (0.85), leading to a strong F1-score (0.85). These metrics indicate that the pipeline is effective at identifying true character names while minimizing false positives and negatives.

5- Co-occurrence matrix

To understand the relationships between characters, a co-occurrence analysis is performed by calculating how often two characters appear together within a certain window of words. This can be done using a sliding window approach, which moves through the text and counts co-occurrences within the window. This step is essential to quantify the strength of relationships between characters.

To create a co-occurrence matrix for each chapter, these steps have been done:

• Extract character names from each chapter as it is done before.

• Store character names in a set to avoid duplicates.

• Create an empty co-occurrence matrix for each chapter.

• For each pair of characters, count the number of times they co-occur within a specific window of sentences.

• Fill the co-occurrence matrix with the counts.

The window size determines the number of consecutive sentences considered for character co-occurrences. A larger window size (e.g., 3) will capture more distant relationships, potentially identifying weaker connections between characters. However, this could also result in more noise. A smaller window size (e.g., 1) focuses on stronger relationships where characters appear together in the same or adjacent sentences, providing a more conservative estimate of character interactions. The choice of window size depends on the desired granularity and specificity of the analysis.

To choose the best window_size for automated model, co-occurrence matrix will be created and other analysis for both window_size = 1 and window_size = 3 and their results will be compared. So, these 2 functions as has been created as below:

1. get_character_names(sentences)

It takes a list of sentences as input and extracts unique character names using the NER pipeline, returning a set of character names.

2. get_co_occurrences_matrix(chapter_sentences, character_names, window_size)

It calculates a co-occurrence matrix for character pairs within a specified window of sentences, taking the chapter sentences, character names, and window size as input, and returning the co-occurrence matrix.

```
# Window size for checking co-occurrences
window_size = 1

def get_character_names(sentences):
    character_names = set()
    for sentence in sentences:
        entities = ner_pipeline(sentence)
        merged_entities = merge_subwords(entities)
        filtered_entities = [entity for entity in merged_entities if is_valid_entity(entity)]
        for entity in filtered_entities:
            character_names.add(entity["word"])
    return character_names

def get_co_occurrences_matrix(chapter_sentences, character_names, window_size):
    name_to_index = {name: i for i, name in enumerate(character_names)}
    co_occurrences = np.zeros((len(character_names), len(character_names)), dtype=int)

    for i in range(len(chapter_sentences) - window_size + 1):
        window_sentences = chapter_sentences[i:i + window_size]
        window_characters = set()
        for sentence in window_sentences:
            entities = ner_pipeline(sentence)
            merged_entities = merge_subwords(entities)
            filtered_entities = [entity for entity in merged_entities if is_valid_entity(entity)]
            for entity in filtered_entities:
                window_characters.add(entity["word"])
        for a, b in combinations(window_characters, 2):
            co_occurrences[name_to_index[a], name_to_index[b]] += 1
            co_occurrences[name_to_index[b], name_to_index[a]] += 1

    return co_occurrences
```

Figure 17

The code below calculates the co-occurrence matrix for all chapters in the text. It first combines sentences from all chapters, and then uses the get_character_names() and get_co_occurrences_matrix() functions to compute the co-occurrence matrix. Finally, it prints the matrix and reports the time taken to create it in minutes.

```
t0=time()
chapter_co_occurrence_matrices = []

all_chapters_sentences = []
for sentences in chapters_sentences:
    all_chapters_sentences.extend(sentences)

print("Processing combined chapters")
character_names = get_character_names(all_chapters_sentences)
co_occurrences = get_co_occurrences_matrix(all_chapters_sentences, character_names, window_size)
chapter_co_occurrence_matrices.append(co_occurrences)
print("Co-occurrence matrix for the combined chapters:")
print(co_occurrences)
print()
t1=time()
print("The time it took to create the co-occurrence matrix in minutes: ", (t1-t0)/60)
```

Figure 18

```
Processing combined chapters
Co-occurrence matrix for the combined chapters:
[[0 0 0 ... 0 0 5]
 [0 0 0 ... 0 0 2]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [5 2 0 ... 0 0 0]]

The time it took to create the co-occurrence matrix in minutes:  30.391440053780872
```

window_size = 1

Processing time : 30 minutes

```
co_occurrences.shape
```

```
(107, 107)
```

Figure 19

```
Processing combined chapters
Co-occurrence matrix for the combined chapters:
[[0 0 3 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [3 0 0 ... 0 0 3]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 3 ... 0 0 0]]

The time it took to create the co-occurrence matrix in minutes:  78.27314721743265
```

window_size = 3

Processing time : 78 minutes

```
co_occurrences.shape
```

```
(107, 107)
```

Figure 20

6- Network graph (`G`)

6-1: Creating the network graph (`G`)

This code creates a network graph (`G`) using the NetworkX library to represent the relationships between characters in the text. First, it adds nodes to the graph for each character's name. Then, it adds edges between character pairs with weights based on their co-occurrence counts from the co-occurrence matrix. Finally, it prints the graph object, which provides information on the number of nodes and edges in the graph.

```
from IPython.display import clear_output
clear_output(wait=True)


G = nx.Graph()    # Create an empty graph

character_names_list = list(character_names) # Convert character_names to a list

G.add_nodes_from(character_names_list)

# Add edges to the graph with weights based on co-occurrence counts
for i in range(len(character_names_list)):
    for j in range(i + 1, len(character_names_list)):
        count = co_occurrences[i][j]
        if count > 0:  # Only add edges with non-zero counts
            G.add_edge(character_names_list[i], character_names_list[j], weight=count)

print(G)
```

Figure 21

Creating the Graph

`window_size =1`

```
print(G)
```

```
Graph with 107 nodes and 360 edges
```

Figure 22

Creating the Graph

`window_size =3`

```
print(G)
```

```
Graph with 107 nodes and 697 edges
```

Figure 23

These results indicate that, for a window size of 1, the graph has 107 character nodes and 360 edges representing their relationships. When the window size is increased to 3, the graph still has 107 character nodes, but the number of edges increases to 697. The increase in edges occurs because a larger window size captures more distant relationships between characters, resulting in a denser graph with more connections.

6-2: Filter out weak connections from network graph:

In the code below, the character network graph is filtered to include just the important characters based on a co-occurrence threshold. First, it calculates the sum of co-occurrences for each character and sets a threshold value. Then, it keeps just the characters whose co-occurrence sum is greater than the threshold, creating a new filtered co-occurrence matrix. Next, it initializes an empty graph (`G_filtered`) and adds nodes for the important characters. Finally, it adds edges between character pairs in the filtered graph with weights based on the filtered co-occurrence counts.

Filter the nodes with weak connections < 3

```
# Calculate the sum of co-occurrences for each character
co_occurrence_sum = np.sum(co_occurrences, axis=1)

threshold = 2

# Keep only characters with a co-occurrence sum greater than the threshold
important_characters_indices = np.where(co_occurrence_sum > threshold)[0]
important_characters = [character_names_list[i] for i in important_characters_indices]

# Create a new co-occurrence matrix with only the important characters
filtered_co_occurrences = co_occurrences[np.ix_(important_characters_indices, important_characters_indices)]
```

```
from IPython.display import clear_output
clear_output(wait=True)

import networkx as nx

G_filtered = nx.Graph() # Create an empty graph

# Add nodes to the graph
G_filtered.add_nodes_from(important_characters)

# Add edges to the graph with weights based on the filtered co-occurrence counts
for i in range(len(important_characters)):
    for j in range(i + 1, len(important_characters)):
        count = filtered_co_occurrences[i][j]
        if count > 0:  # Only add edges with non-zero counts
            G_filtered.add_edge(important_characters[i], important_characters[j], weight=count)
```

Figure 24

```
print(G)
```

```
Graph with 107 nodes and 360 edges
```

```
print(G_filtered)
```

```
Graph with 55 nodes and 320 edges
```

Figure 25

The initial graph contained 107 nodes and 360 edges, representing a relatively dense network of character relationships. After filtering out weak connections (those with a co-occurrence value of 2 or less), a graph consisting of 55 nodes and 320 edges has been remained.

This filtered graph represents a more focused network of character relationships, emphasizing only the stronger connections between characters. By removing weak connections, minor and less significant relationships or some false identified characters by NER are eliminated, allowing for a clearer understanding of the main interactions and communities within the network.

The reduction from 107 nodes to 55 nodes while the majority of edges (320 out of 360) are still present after filtering, indicates that the core structure of the network has been preserved, and the relationships among the central characters remain well-represented.

6-3: Visualizes the filtered network graph

This code visualizes the filtered character network graph (`G_filtered`). The nx.draw() function is used to draw the graph with specified attributes such as node size, font size, node color, edge color, and edge width. The graph is drawn with node labels (character names). Edge labels represent the co-occurrence count between character pairs. The edge labels are created using a dictionary comprehension and drawn using nx.draw_networkx_edge_labels() function with specified font size.

```
# Draw the graph
pos = nx.spring_layout(G_filtered, k=9)
fig, ax = plt.subplots(figsize=(15, 10))  # Increase the figure size
nx.draw(G_filtered, pos, with_labels=True, node_size=4000, font_size=12, node_color="skyblue",
        edge_color="gray", width=3, ax=ax)
edge_labels = {(important_characters[i], important_characters[j]): filtered_co_occurrences[i][j]
               for i in range(len(important_characters)) for j in range(i + 1, len(important_characters))
               if filtered_co_occurrences[i][j] > 0}
nx.draw_networkx_edge_labels(G_filtered, pos, edge_labels=edge_labels, font_size=10, ax=ax)

plt.savefig("G.png", dpi=300, bbox_inches="tight")
plt.show()
```

Figure 26

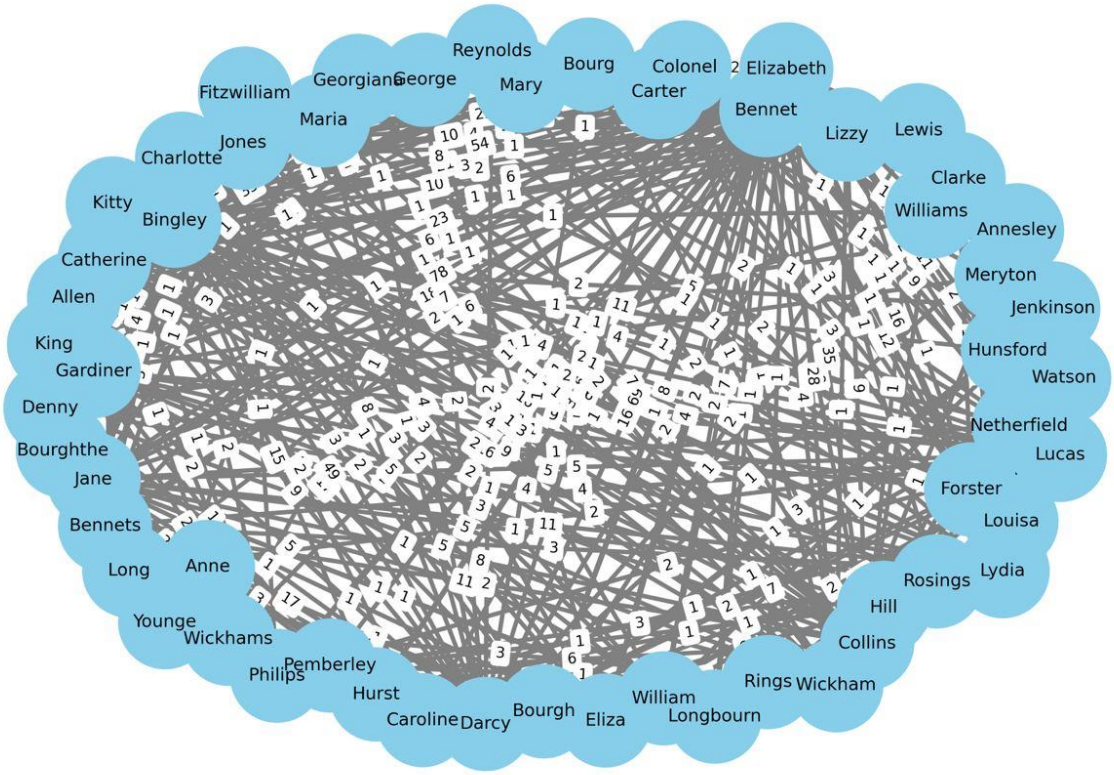


Figure 27

7- Centrality measurements

• Degree Centrality: Degree centrality measures the number of direct connections a character has. Characters with high degree centrality are well-connected and have more relationships with other characters.

• Closeness Centrality: Closeness centrality measures how close a character is to all other characters in the network. Characters with high closeness centrality can quickly interact or influence other characters.

• Betweenness Centrality: Betweenness centrality measures the extent to which a character lies on the shortest path between other characters. Characters with high betweenness centrality act as bridges or intermediaries within the network.

• Eigenvector Centrality: Eigenvector centrality measures the influence of a character based on the quality of their connections. Characters with high eigenvector centrality have connections with other influential characters.

In the code below, centrality measurements are calculated for filtered graph G.

```
degree_centrality = nx.degree_centrality(G_filtered)
closeness_centrality = nx.closeness_centrality(G_filtered)
betweenness_centrality = nx.betweenness_centrality(G_filtered)
eigenvector_centrality = nx.eigenvector_centrality(G_filtered)
```

Figure 28

The following code creates a bar chart to visualize four centrality measures (Degree, Closeness, Betweenness, and Eigenvector) for each character in the graph.

```
characters = list(degree_centrality.keys())
degree_values = list(degree_centrality.values())
closeness_values = list(closeness_centrality.values())
betweenness_values = list(betweenness_centrality.values())
eigenvector_values = list(eigenvector_centrality.values())

# Set up the bar chart
num_characters = len(characters)
bar_width = 0.2
x_pos = np.arange(num_characters)

plt.figure(figsize=(20, 10), dpi=300)
plt.bar(x_pos, degree_values, bar_width, alpha=0.5, color='r', label='Degree_centrality')
plt.bar(x_pos + bar_width, closeness_values, bar_width, alpha=0.5, color='g', label='Closeness_centrality')
plt.bar(x_pos + 2 * bar_width, betweenness_values, bar_width, alpha=0.5, color='b', label='Betweenness_centrality')
plt.bar(x_pos + 3 * bar_width, eigenvector_values, bar_width, alpha=0.5, color='y', label='Eigenvector_centrality')

plt.xticks(x_pos + bar_width, characters, rotation=90, fontsize=10)
plt.xlabel('Characters')
plt.ylabel('Centrality')
plt.title('Centrality Measures for Characters')
plt.legend()
plt.savefig("centrality_measures_w1.png", dpi=300, bbox_inches="tight")
plt.show()
```

Figure 29

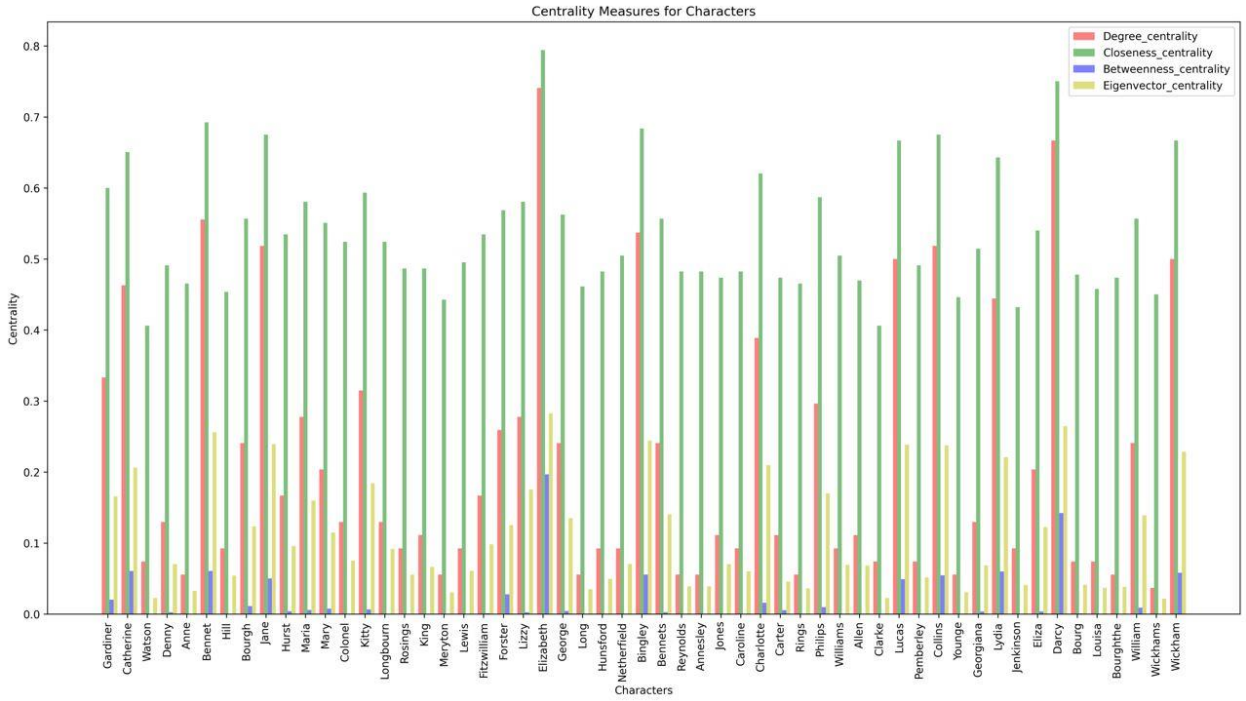


Figure 30

The bar chart compares the centrality of characters in the network based on four different metrics. Degree centrality indicates the number of direct connections a character has, while closeness centrality represents how close a character is to all other characters in the network. Betweenness centrality measures the extent to which a character acts as a bridge or intermediary within the network, and eigenvector centrality takes into account the importance of a character's neighbors in the network. By analyzing these centrality measures, we can identify the most influential or central characters in the network and better understand their roles within the story.

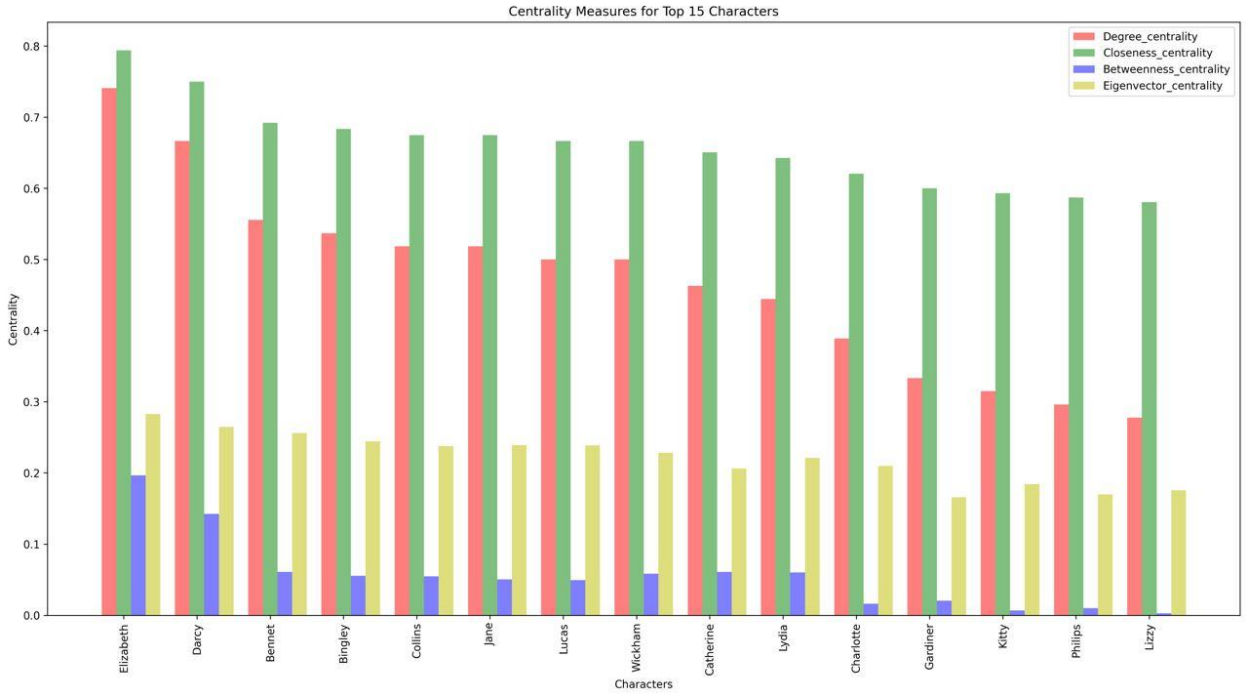


Figure 31

Degree Centrality: Elizabeth, Darcy, and Bennet have the highest degree centrality, which suggests that they have the most significant number of direct connections with other characters.

Closeness Centrality: Elizabeth and Darcy have the highest closeness centrality, indicating that they can easily communicate or reach other characters in the network.

Betweenness Centrality: Elizabeth has the highest betweenness centrality, suggesting that she plays a crucial role in connecting different groups or sub-networks of characters.

Eigenvector Centrality: The eigenvector centrality values are relatively close for the top 15 characters. This suggests that the characters are connected with similarly influential characters, and their overall influence in the network is not vastly different.

```
centrality_df.head(15)
```

	Character	Degree	Closeness	Betweenness	Eigenvector
0	Elizabeth	0.740741	0.794118	0.196639	0.282788
1	Darcy	0.666667	0.750000	0.142326	0.264769
2	Bennet	0.555556	0.692308	0.060788	0.256065
3	Bingley	0.537037	0.683544	0.055476	0.244504
4	Jane	0.518519	0.675000	0.050343	0.239325
5	Collins	0.518519	0.675000	0.054659	0.237754
6	Lucas	0.500000	0.666667	0.049251	0.238630
7	Wickham	0.500000	0.666667	0.058252	0.228477
8	Catherine	0.462963	0.650602	0.060693	0.206329
9	Lydia	0.444444	0.642857	0.059977	0.221022
10	Charlotte	0.388889	0.620690	0.016070	0.209767
11	Gardiner	0.333333	0.600000	0.020373	0.165788
12	Kitty	0.314815	0.593407	0.006603	0.184144
13	Philips	0.296296	0.586957	0.009891	0.169989
14	Maria	0.277778	0.580645	0.005693	0.160132

Figure 32

The table above presents the centrality measures for the top characters in the network. Elizabeth has the highest degree, closeness, betweenness, and eigenvector centrality, indicating she is the most central and influential character in the story. Darcy follows as the second most important character, with high centrality values in all metrics. Other characters, such as Bennet and Bingley, also have notable centrality measures, suggesting they play significant roles in the narrative.

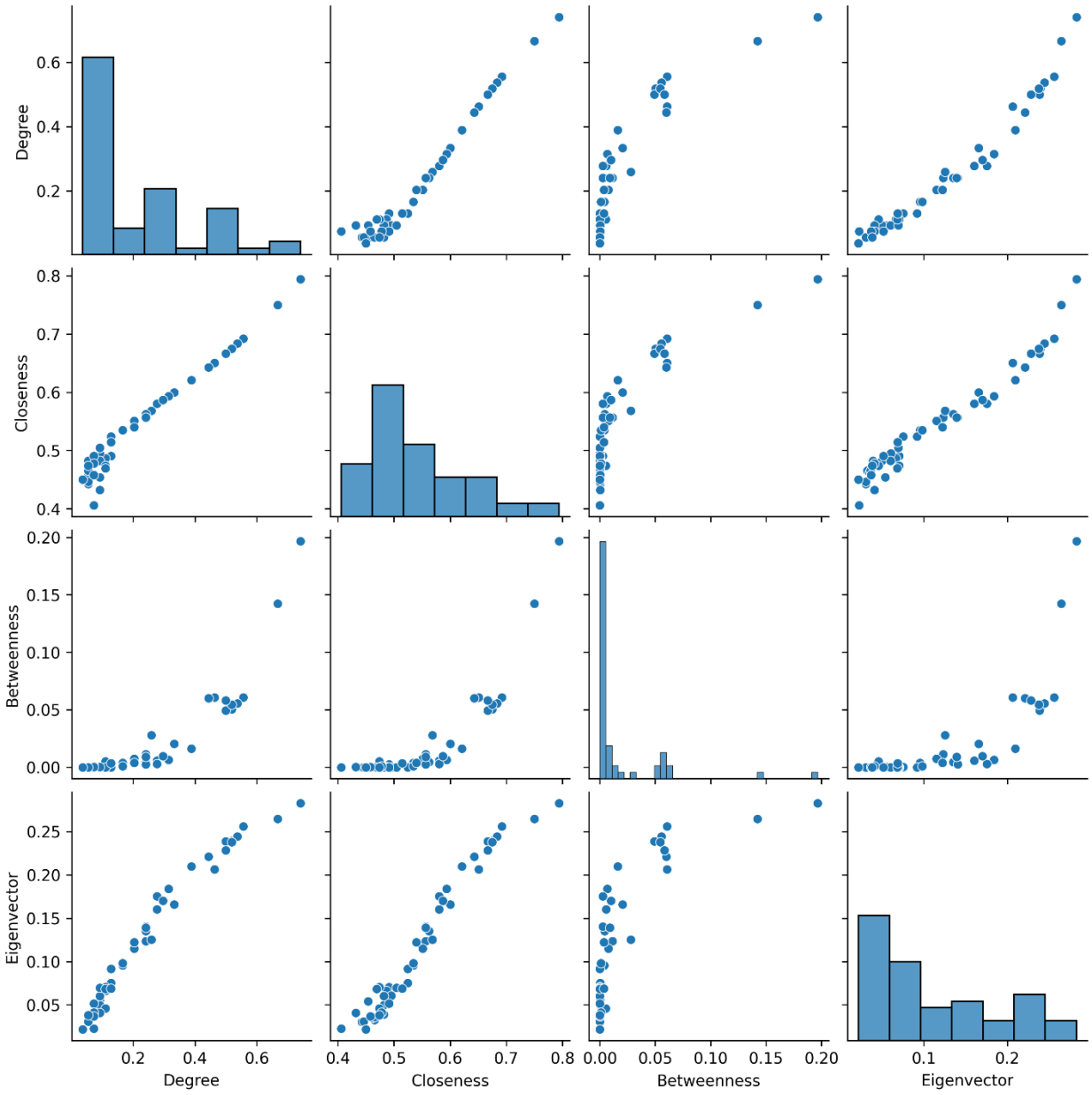


Figure 33

A scatter plot matrix shows the relationship between pairs of centrality measures for all characters in the network. To interpret the scatter plot matrix, we can look for the following patterns:

1. Positive correlation: If two centrality measures have a positive correlation, it means that as one measure increases, the other measure also tends to increase. In the context of a story, this could indicate that characters with high importance in one aspect of the network are also likely to be important in another aspect. For example, when degree centrality and eigenvector centrality are positively correlated, it suggests that well-connected characters are also connected to other influential characters.

2. Negative correlation: If two centrality measures have a negative correlation, it means that as one measure increases, the other measure tends to decrease. This could reveal that characters with high importance in one aspect of the network are less important in another aspect.

3. No correlation: If two centrality measures do not show any clear pattern or relationship, it means that they are not correlated. This suggests that the importance of characters in one aspect of the network is independent of their importance in another aspect. For example, a character with high closeness centrality may not necessarily have high betweenness centrality, like Lizzy(one of 15 most important character).

4. Outliers: Outliers are data points that stand apart from the general trend or pattern. In the context of a story, outliers could represent unique characters who play a distinctive role in the network. For example, a character with a very high betweenness centrality but low degree centrality could be an essential mediator or bridge between different groups, even though they have few connections.

When interpreting a scatter plot matrix, it's essential to consider the context of the story and the meaning behind each centrality measure. By understanding the relationships between centrality measures and identifying outliers, we can gain valuable insights into the characters' roles and their influence on the story's network.

8- Network Analysis:

8-1. Degree Distribution:

Does our network follow a power-law distribution?

Degree distribution is a concept used in network analysis to describe the probability distribution of the degrees (number of connections) over all the nodes (entities) in a network. In simple terms, it is a way to represent how many connections each node in the network has and how these connections are distributed among all the nodes.

In the context of novels study, the degree distribution represents how character relationships are distributed within the social network formed by the characters. A high degree for a character node means that the character has many connections (relationships) with other characters, while a low degree indicates fewer connections. By analyzing the degree distribution of the networks, we can gain insights into the structure of character relationships and the prominence of certain characters in the novels.

A power-law distribution is a type of probability distribution that exhibits a specific mathematical relationship between the quantity being measured (in this case, the degree of nodes in a network) and the frequency of that quantity.

When a network is said to follow a power-law distribution, it implies that the network has a few highly connected nodes, called "hubs," while the majority of the nodes have relatively fewer connections. This property is often observed in real-world networks, such as social networks, where some individuals have a large number of connections while most have only a few connections.

```
# Degree Distribution
degree_values = [v for k, v in degree_centrality.items()]  # degree_values is equal to degree_centrality
degree_distribution = np.histogram(degree_values, bins=len(set(degree_values)), density=True)
hist, bin_edges = degree_distribution

plt.figure(figsize=(5, 4))
plt.bar(bin_edges[:-1], hist, width=np.diff(bin_edges), edgecolor="black", alpha=0.7)
plt.xlabel("Degree")
plt.ylabel("Frequency")
plt.title("Degree Distribution,  window-size = 1")
plt.savefig("Degree Distribution1.png", dpi=300, bbox_inches="tight")
plt.show()
```

Figure 34

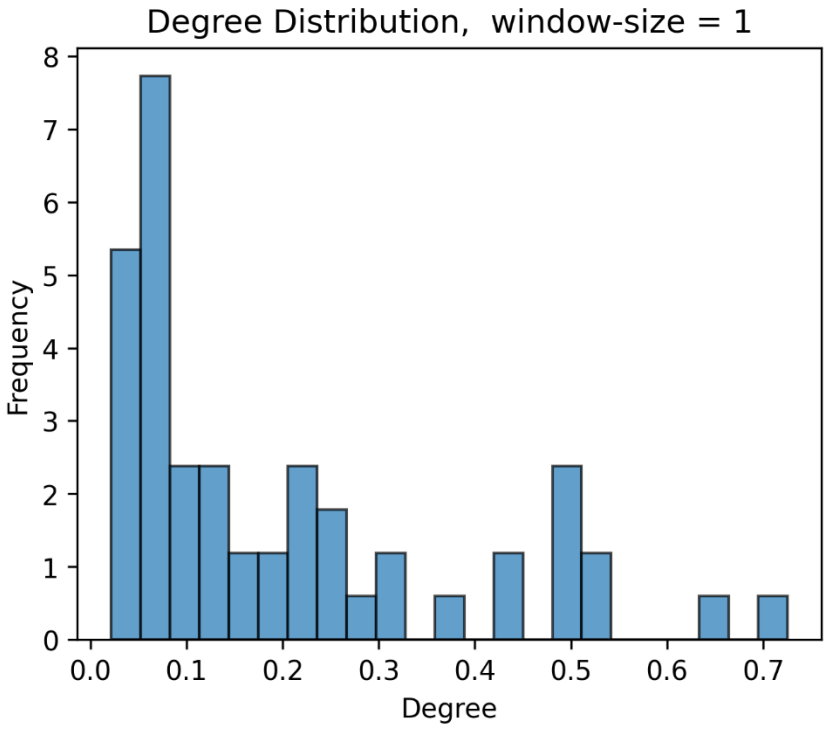


Figure 35

To further confirm that our network follows a power-law distribution, a linear model is fitted to the log-log data and the goodness-of-fit, such as the R-squared value is computed. A high R-squared value (close to 1) indicates that the power-law model fits our data well, providing stronger evidence that our network follows a power-law distribution or not.

```python
# Logarithmic binning
bins = degree_distribution[1]
bin_centers = (bins[:-1] + bins[1:]) / 2
hist_values = degree_distribution[0]

# Remove zero values for log-log plotting
non_zero_indices = hist_values > 0
bin_centers = bin_centers[non_zero_indices]
hist_values = hist_values[non_zero_indices]

# Fit a linear model
log_centers = np.log10(bin_centers)
log_values = np.log10(hist_values)
coefficients = np.polyfit(log_centers, log_values, 1)
linear_fit = np.poly1d(coefficients)

# Calculate R-squared
correlation_matrix = np.corrcoef(log_centers, log_values)
correlation_coefficient = correlation_matrix[0, 1]
r_squared = correlation_coefficient ** 2

# Log-log plot with fitted line
plt.plot(bin_centers, hist_values, 'o', label='Data')
plt.plot(bin_centers, 10 ** linear_fit(log_centers), label='Fitted Line')
plt.xscale('log')
plt.yscale('log')
plt.xlabel('Degree (normalized)')
plt.ylabel('Probability Density')
plt.title(f'Degree Distribution (Log-Log Plot)\nR-squared = {r_squared:.2f}')
plt.legend()
plt.savefig("Degree Distribution.png", dpi=300, bbox_inches="tight")
plt.show()
```

Figure 36

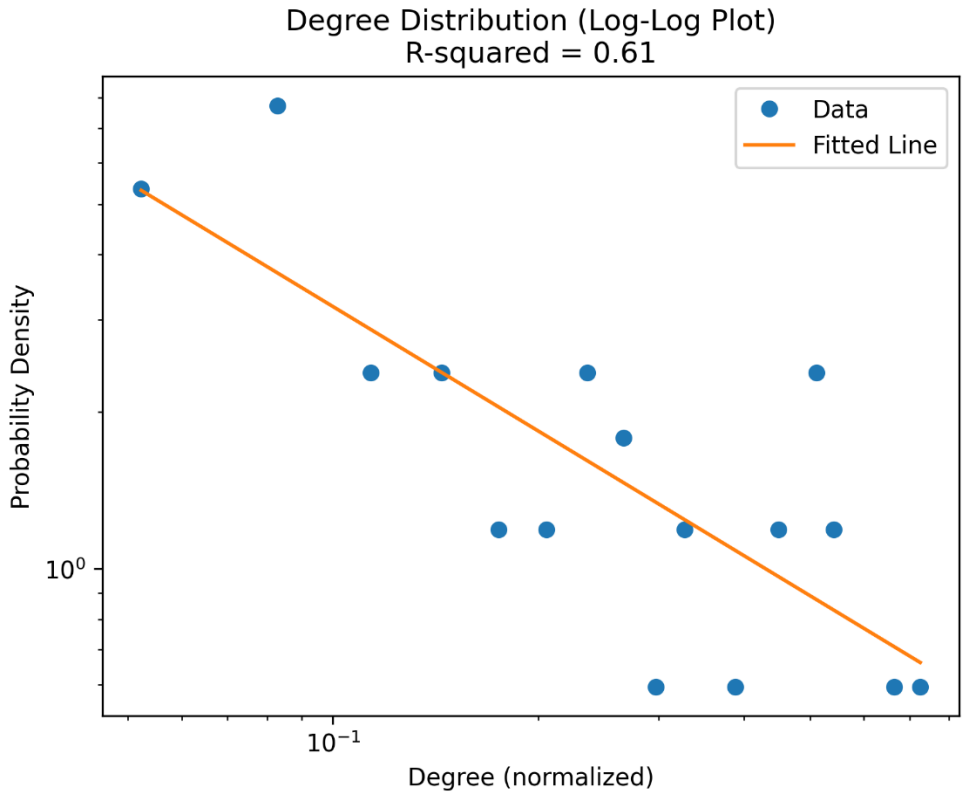


Figure 37

In a log-log plot, the x-axis represents the degree (normalized) and the y-axis represents the probability density of that degree occurring in the network. A negative linear relationship implies that as the degree increases, the probability density of that degree occurring in the network decreases, But the value of R-squared is 0.61, which suggests that this network does not follow a power-law distribution strongly.

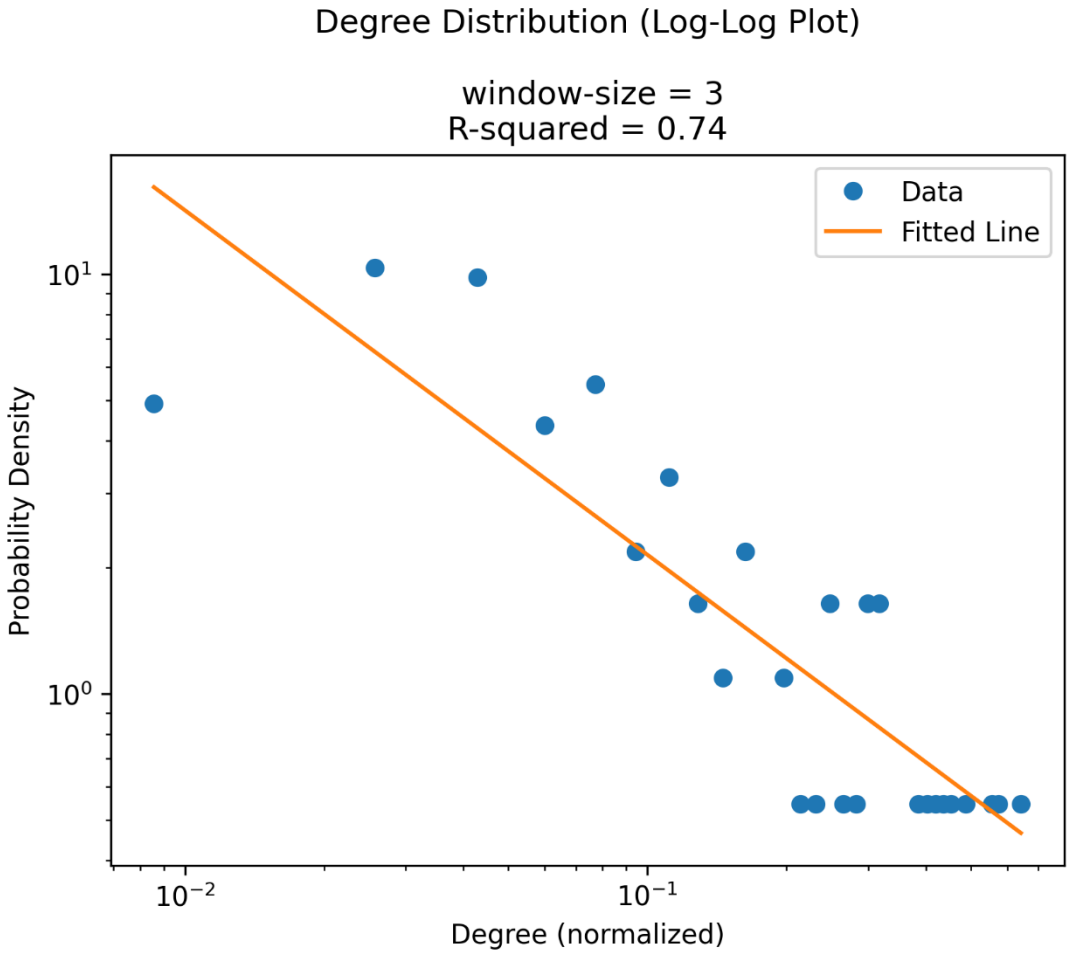


Figure 38

On the other hand, an R-squared value of 0.74 with a window size of 3 indicates that the power-law model explains 73% of the variance in your log-log degree distribution data. This is a lower goodness-of-fit compared to the window size of 1, but it suggests that our network exhibits power-law characteristics to some extent.

The difference in R-squared values between the window sizes implies that the choice of window size affects the degree distribution and the power-law property of my network. A larger window size might capture more local character relationships within sentences, leading to a stronger power-law property, while a smaller window size might capture more global character relationships across sentences, which could dilute the power-law property to some extent.

8-2: Network density

Network density is a measure of the overall connectivity of a network. It represents the proportion of actual edges in the network to the maximum possible number of edges between all nodes. A high network density indicates that the network is highly interconnected, while a low network density suggests that the network is sparse.

8-3: Component size, Average Shortest Path Length, Diameter

```
# Find the connected components
connected_components = list(nx.connected_components(G_filtered))

# Initialize lists to store data for the DataFrame
component_nodes = []
component_sizes = []
average_shortest_path_lengths = []
diameters = []

# Calculate the average shortest path length and diameter for each connected component
for component in connected_components:
    subgraph = G.subgraph(component)
    average_shortest_path_length = nx.average_shortest_path_length(subgraph)
    diameter = nx.diameter(subgraph)

    component_nodes.append(list(component))
    component_sizes.append(subgraph.number_of_nodes())
    average_shortest_path_lengths.append(average_shortest_path_length)
    diameters.append(diameter)

components_df = pd.DataFrame(
    {
        "Component Nodes": component_nodes,
        "Component Size": component_sizes,
        "Average Shortest Path Length": average_shortest_path_lengths,
        "Diameter": diameters,

    }
)
```

Figure 39

```
window-size = 1
```

	Component Nodes	Component Size	Average Shortest Path Length	Diameter
0	[Williams, Bennets, Gardiner, Catherine, Allen...	55	1.901684	3

Figure 40

```
window-size = 3
```

	Component Nodes	Component Size	Average Shortest Path Length	Diameter
0	[Forster, Darcys, Lucas, Netherfield, Wickhami...	103	2.071007	4
1	[Nichols]	1	0.000000	0
2	[Grantley]	1	0.000000	0
3	[Stone]	1	0.000000	0
4	[Webb]	1	0.000000	0

Figure 41

In a network analysis of two different window sizes, 1 and 3, we observe the following characteristics for the largest connected components:

Window-Size 1:

• Component Size: The largest connected component comprises 55 nodes, indicating that there are 55 nodes connected to at least one other node in the network.

• Average Shortest Path Length: The average shortest path length within the largest connected component is 1.901684, suggesting that it takes approximately 1.9 steps, on average, to traverse from one node to another in this component.

• Diameter: With a diameter of 3, the longest shortest path between any two nodes in the largest connected component spans 3 steps.

Window-Size 3:

• Component Size: In this case, the largest connected component contains 103 nodes, signifying those 103 nodes are connected to at least three other nodes in the network. This larger component, when compared to window-size 1, implies increased connectivity among nodes as the window size expands.

• Average Shortest Path Length: The average shortest path length experiences a slight increase to 2.071007, indicating that it takes roughly 2.07 steps to navigate from one node to another within the largest connected component.

• Diameter: The diameter of the largest connected component grows to 4, revealing that the longest shortest path between any two nodes in the component now extends to 4 steps.

So as the window size increases from 1 to 3, the largest connected component becomes more extensive, and both the average shortest path length and diameter exhibit growth. These findings suggest that the network's interconnectivity and complexity are heightened as the window size increases. The term "longest shortest path" refers to the greatest distance between any two nodes in a network, measured by the number of steps or connections between them, following the shortest possible route. In other words, it represents the maximum number of steps required to travel from one node to another using the most direct path.

8-4: Clustering coefficient

```python
clustering_coefficient = nx.clustering(G_filtered)

average_clustering_coefficient = nx.average_clustering(G_filtered)

print(f"Average Clustering Coefficient: {average_clustering_coefficient}")

# Create a DataFrame from the clustering_coefficient dictionary
clustering_coefficient_df = pd.DataFrame.from_dict(
    clustering_coefficient, orient='index', columns=['Clustering Coefficient']
)

# Sort the DataFrame by Clustering Coefficient in descending order (large to small)
sorted_clustering_coefficient_df = clustering_coefficient_df.sort_values(
    by=['Clustering Coefficient'], ascending=False
)

# Round the values in the DataFrame to 2 decimal places
rounded_sorted_clustering_coefficient_df = sorted_clustering_coefficient_df.round(2)
```

Figure 42

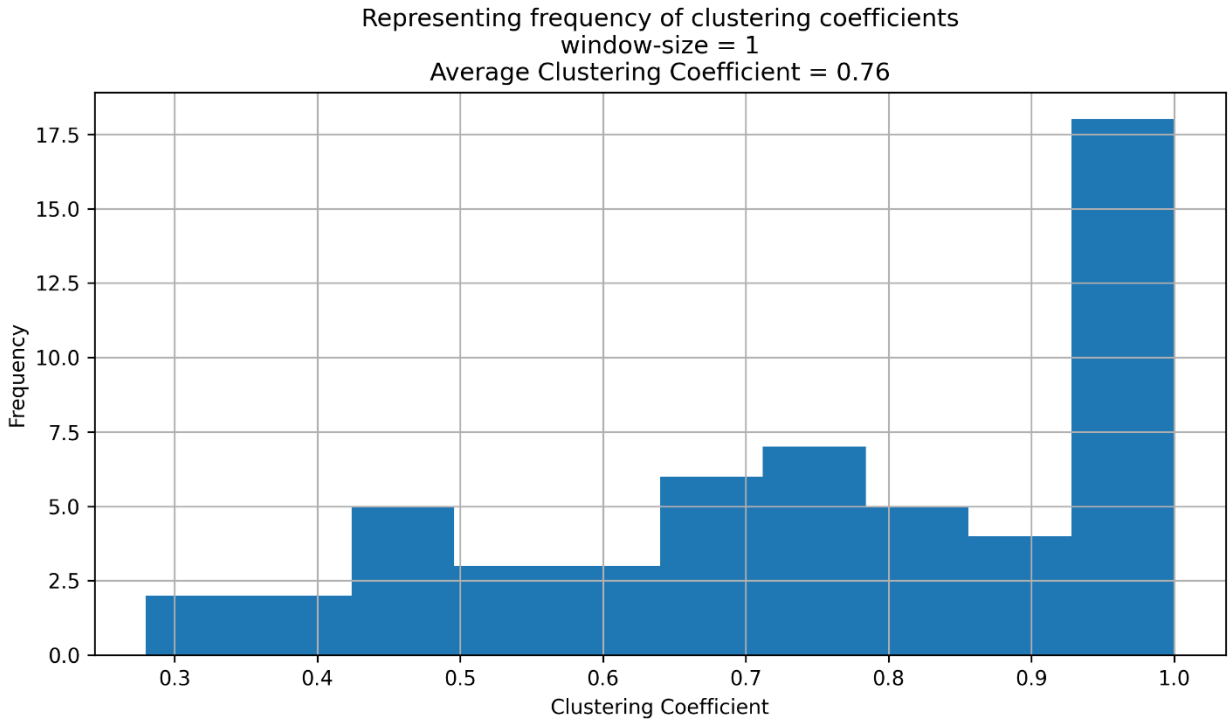


Figure 43

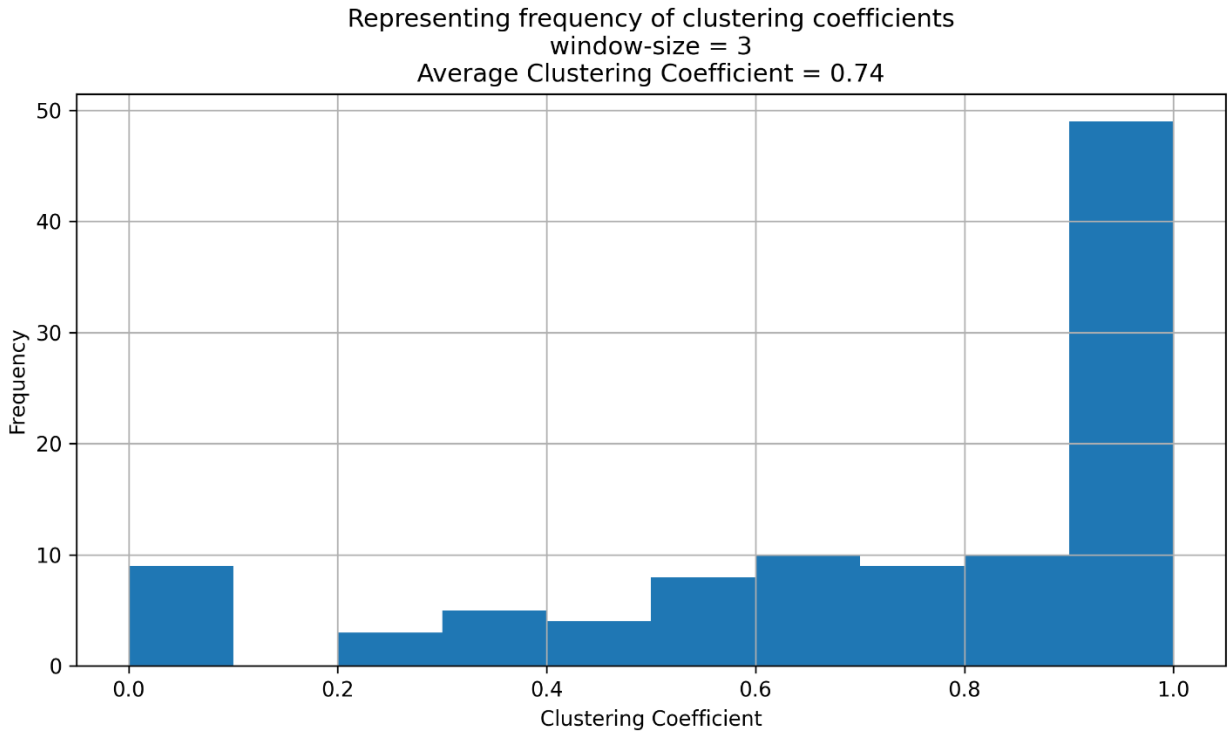


Figure 44

Window size	Number of nodes	Number of edges	Degree-distribution R^2	Density	Number of connected components	Average shortest path	Average diameter	Average clustering coefficient	Time (minute)
1	**107**	**360**	**0.9**	**0.02**	**84**	**2.20**	**4.69**	**0.5**	**30**
3	**107**	**697**	**0.74**	**0.12**	**103**	**2.07**	**4**	**0.74**	**78**

Figure 45

Based on the all the results and their comparisons and regarding the table above, the more reasonable approach is to use window size=1 to create the automatic model, since it is less time-consuming. So, the community detection will be applied here just for window size=1.

9- Community detection

Community detection is a technique used to identify groups or clusters of nodes in a network that are more densely connected with each other than with nodes outside the group. Girvan-Newman algorithm is used for community detection. By applying Girvan-Newman algorithm to the network, we can identify clusters of characters that are more closely related. This can help to understand different groups or factions within the story and reveal interesting patterns or social groups in the story.

9-1: Girvan-Newman algorithm

The Girvan-Newman algorithm is a community detection method in network analysis. It identifies communities by iteratively removing edges with high betweenness centrality, which are the edges

connecting different clusters. This process continues until the network is divided into distinct, well-defined communities. So, in the figure below, community detection for the whole network and for characters which have more than two connections, is shown.

```
import networkx as nx
from networkx.algorithms.community import girvan_newman

communities_generator = girvan_newman(G)
# Get the first level of communities (I can call next() again for more refined communities)
communities = next(communities_generator)
# The communities variable is a list of sets, each set representing a community
print(communities)
```

Figure 46

```
# Function to map colors to the nodes
def map_colors(node, communities):
    for i, community in enumerate(communities):
        if node in community:
            return i
    return -1

# Create a new figure
plt.figure()

# Set the position of nodes using a spring layout
pos = nx.spring_layout(G)

# Draw the nodes with different colors for different communities
nx.draw(G, pos, node_color=[map_colors(node, communities) for node in G.nodes()], with_labels=True)

# Draw the edges
nx.draw_networkx_edges(G, pos, alpha=0.5)

plt.title('Clustering, Communities of all characters ')
plt.savefig("Clustering, all Communities.png", dpi=300, bbox_inches="tight")
plt.show()
```

Figure 47

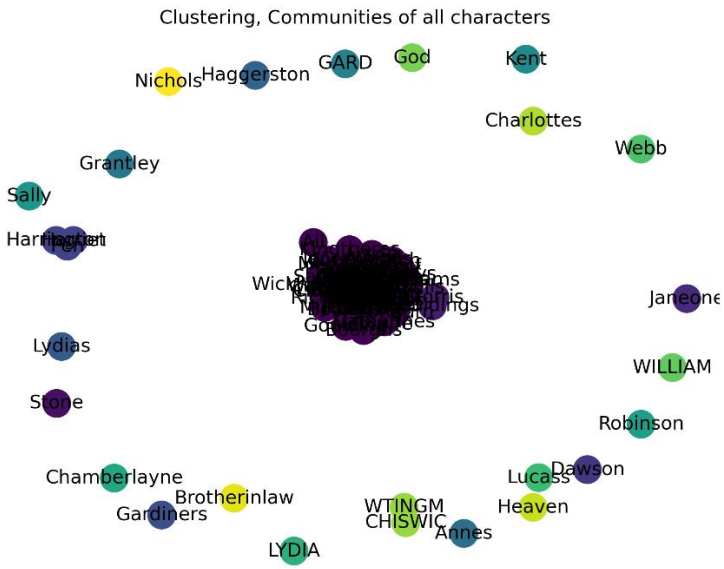


Figure 48

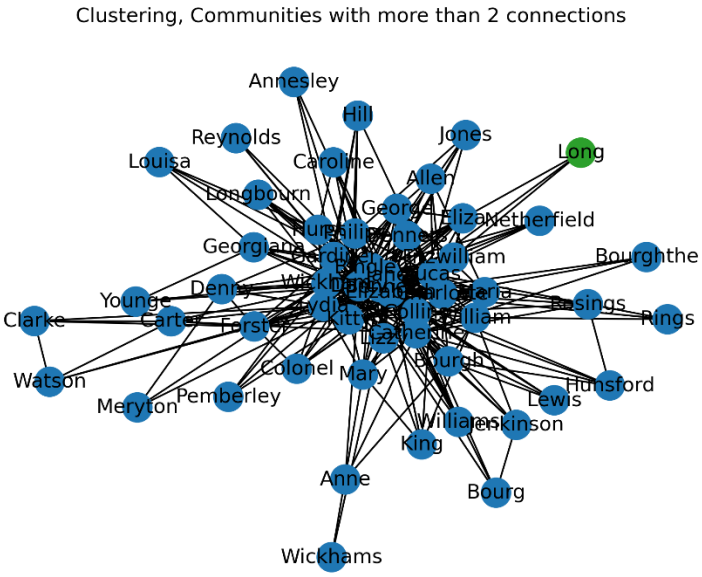


Figure 49

Characters	Number of nodes	Number of edges	Degree-distribution R^2	Density	Number of connected components	Average shortest path	Average diameter	Average clustering coefficient
All	**107**	**360**	**0.9**	**0.02**	**84**	**2.20**	**4.69**	**0.76**
Connections >2	**55**	**320**	**0.61**	**0.21**	**55**	**1.9**	**3**	**0.74**

Figure 50

Hence, filtering the network to the characters with more than 2 connections is a proper approach to remove the outliers and errors from NER stage. Since after filtering, even though the number of nodes is decreased significantly from 107 to 55, the number of edges is preserved, just a reduction from 360 to 320 connections. This suggests that by filtering the network in preserved and connections has not been lost. To selected approach is to create the automatic model and removing the outliers, as it is not possible to check each single book during the analysis.

Conclusion

In conclusion, the analysis of character relationships in "Pride and Prejudice" revealed that using a window size of 1 is a more efficient and reasonable approach for creating an automatic model, as it reduces computation time. The community detection was applied to this window size, and by filtering the network to include characters with more than 2 connections, the model effectively removed outliers and errors from the NER stage. Although the filtering process significantly reduced the number of nodes from 107 to 55, the majority of the edges were preserved, with only a slight reduction from 360 to 320 connections. This suggests that the core structure of the network and the relationships among central characters were maintained. The selected approach provides a valuable method for creating an automatic model and removing outliers, offering a scalable

solution for analyzing character relationships in various literary works without requiring manual intervention.

References

Benkassioui, B., Kharmoum, N., Hadi, M.Y. and Ezziyyani, M., 2022, May. NLP Methods' Information Extraction for Textual Data: An Analytical Study. In *International Conference on Advanced Intelligent Systems for Sustainable Development* (pp. 515-527). Cham: Springer Nature Switzerland.

Brahman, F., Huang, M., Tafjord, O., Zhao, C., Sachan, M. and Chaturvedi, S., 2021. " Let Your Characters Tell Their Story": A Dataset for Character-Centric Narrative Understanding. *arXiv preprint arXiv:2109.05438*.

Chaturvedi, S., Srivastava, S., Daume III, H. and Dyer, C., 2016, March. Modeling evolving relationships between characters in literary novels. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 30, No. 1).

Chaturvedi, S., Iyyer, M. and Daume III, H., 2017, February. Unsupervised learning of evolving relationships between literary characters. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 31, No. 1).

Devisree, V. and Raj, P.R., 2016. A hybrid approach to relationship extraction from stories. *Procedia Technology*, *24*, pp.1499-1506.

Fernandez, M., Peterson, M. and Ulmer, B., 2015. Extracting social network from literature to predict antagonist and protagonist. *Recuperado de: https://nlp. stanford. edu/courses/cs224n/2015/reports/14. pdf*.

https://www.gutenberg.org

https://github.com/ChimdinduDenalexOrakwue/Character-Relationship-Analysis?search=1

https://github.com/isthatyoung/NLP-Characters-Relationships/tree/master

Inoue, N., Pethe, C., Kim, A. and Skiena, S., 2022, May. Learning and Evaluating Character Representations in Novels. In *Findings of the Association for Computational Linguistics: ACL 2022* (pp. 1008-1019).

Krug, M., 2020. *Techniques for the Automatic Extraction of Character Networks in German Historic Novels*. Bayerische Julius-Maximilians-Universitaet Wuerzburg (Germany).

Massey, P., Xia, P., Bamman, D. and Smith, N.A., 2015. Annotating character relationships in literary texts. *arXiv preprint arXiv:1512.00728*.

Nijila, M. and Kala, M.T., 2018, July. Extraction of Relationship Between Characters in Narrative Summaries. In *2018 International Conference on Emerging Trends and Innovations In Engineering And Technological Research (ICETIETR)* (pp. 1-5). IEEE.

Nundloll, V., Smail, R., Stevens, C. and Blair, G., 2022. Automating the extraction of information from a historical text and building a linked data model for the domain of ecology and conservation science. *Heliyon*.

Yan, R., Jiang, X., Wang, W., Dang, D. and Su, Y., 2022. Materials information extraction via automatically generated corpus. *Scientific Data*, *9*(1), p.401.