Mark McGown

10021 N Polk Ave
Portland, OR, 97203
(503) 200-8987
markamcgown@gmail.com

Data Engineer with 8 years of experience & MSDS. Aiming to leverage my skills to successfully fill Senior Database Analyst role.

PROFESSIONAL EXPERIENCE

INTEL CORP, Hillsboro, OR

Senior Database Analyst | Mar 2016 - Present

- Engineered Data Analytics insight via Elasticsearch (ELK) by building pipelines, establishing statistical framework, and connecting Python/R.
- Conducted regression analysis in Python & MS SQL for models of DB performance.
- Develop highly efficient database design for web applications serving the constraints of the Team Lead, IT, and the customer's usage behavior.
- Lead team of vendors & consultants to incorporate new DB capabilities that serve the Machine Learning environment.

INTEL CORP, Phoenix, AZ

Database Analytics Engineer | Aug 2011 - Mar 2016

- Programmed automated SQL/JMP data pulls to identify process defects & customer latency analytics. Presented findings to Engineers/PM's.
- Built ML predictive models to estimate customer behavior leading to safe assumptions. Layered complex scripts on thesis of customer behavior.
- Predictive modeled recommendation systems to feedback to automation systems. Throttled/released production accordingly on future insights.

EDUCATION

NORTHWESTERN UNIVERSITY

Evanston, IL

M.S. Data Science (Jun 2020)

- **GPA:** 4.0
- **Award And Honor:** Magna Cum Laude

OHIO STATE UNIVERSITY

Columbus, OH

B.S. Chemical Engineering (Jun 2011)

REFERENCES

DON LONG, Systems Engineer, Intel Corp

(503) 209-0706 - Coworker

BRIAN OUELLETTE, Software Engineer, Intel Corp

(704) 957-5853 - Mentor

The Networks of Co-Occurring Proper Nouns In Anne Frank's Diary

By Mark McGown

Abstract

This work aimed to examine Anne Frank's *The Diary of a Young Girl* (Bantam Books 1993) from the perspective of network science. Data was first collected from The Internet Archive as a raw text file, and sentiment as well as proper nouns were extracted using the NLTK and spaCy libraries in Python. The relationships of sententially co-occurring proper nouns in the overall text were then compiled and analyzed using the NetworkX module in Python. Community partitioning, assortativity analysis, ego-centric subgraphs, cliques, and centrality indices of the networks of proper nouns were studied in order to discover the structural entanglement and clustering of co-occurrent terms. Communities were discovered amongst these proper nouns that showed correlation with the relationships between them. Significant shifts in sentiment both between community partitions as well as temporally were found. The role of each proper noun was found to correlate with its relative ranking in specific measures of centrality. This analysis suggested that data science and network analysis are adept tools at aggregating meaning from text documents. The findings of this project would be of interest to scholars of the humanities and digital humanities, data scientists studying text mining and text analytics, social scientists and psychologists working on biographical narrative interpretive methods, and historians of WWII intellectual and social events.

Table of Contents

I.Introduction

Anne Frank's diary presents the perspective of a world shifting into conflict during World War II from the eyes of a 13-year-old Jewish girl. It is as compelling as a first-hand account of a religious group of people facing extermination as it is innocent in the eyes of an adolescent. Rarely in modern history does an account of so many disparate communities, sentiment, and political forces emerge to tell a story of conflict, heroism, and tragedy. The contents of this diary blend the day-to-day interests of a German adolescent hiding in Amsterdam with the unfolding events of a German-occupied Netherlands. The context of how each character relates to one another is open to the interpretation of the reader. However, this is only subjective and contingent on the reader's prior knowledge.

Since such interpretations cannot make accurate statements about the entire text, the goal of this research was to reduce subjectivity by aggregating the relationships of proper nouns. The proper nouns in this analysis will include not just characters from the diary but also the warring nations and organizations impacting those characters. Aggregations differ from interpretation by averaging the sentiment observed when any pair of proper nouns co-occur. Further, they enable the study which co-occurring proper nouns belong in communities together. These communities themselves can then be studied as they relate to one another, both with the average sentiment between them as well as how this average sentiment changes over time.

The motivation behind this research is two-fold. First, future research should employ these tools to analyze other historical texts as well as compare/contrast differences from Anne Frank's experience. Second, specific findings regarding the aggregation of sentiment may be

instructive in the humanitarian intervention of persecution or nations shifting into instability. The creation of tools that equally instruct as much about history as they do about warning signs in future events serves the foundation for this project.

In order to accomplish the goals of this research, the problem statement guiding this experiment will first be given. As it relates to the need for a reduction in interpretation and ambiguity, the justification for this research and network science, in general, will then be provided. A large part of this research came from the much fuller history of the development of these network concepts. For that reason, the literature that contributed to this research will next be reviewed and given context. Illustrations of how significant community detection algorithms work will be presented in this section. Following the literature review, a historical perspective will illustrate the context of the common proper nouns in the diary. Finally, the full network analysis of Anne Frank's diary will be given, followed by conclusions drawn from the analysis. Each conclusion topic will draw findings as they relate to the stated objectives and literature review.

II.Problem Statement

The problem that this work aims to address pertains solely to the relationship of proper nouns in Anne Frank's diary. A proper noun is defined as a single entity referring to that entity as differentiated from a common noun, which refers to entities and may be used when referring to instances of a specific class (Anderson, 2007). As Quirk et al. (1985) argue, proper nouns "lack number contrast, determination, and modification." Entities unable to take on

various forms of meaning throughout aggregation allow accurate conclusions to be drawn about the contents of the diary.

At the start of this experiment, it was unknown if these proper nouns would exhibit a specific form of preferential attachment. This preferential attachment, as literature review will show, would be the same metric that real-world networks exhibit. This theory was measured by plotting the distribution of relationships each node had and seeing if it fit a power-law distribution. It was found that Anne Frank's diary did not have this behavior of preferential attachment like that of real-world networks. Although Anne Frank's use of narrative did not mirror the behavior many real networks exhibit, this does not mean that texts like hers cannot be used to make predictions about real-world networks involving persecution. Instead, it indicates that this author facing persecution wrote in such a way that a randomly distributed network was the result. It also begs the question, do other autobiographical authors write in this manner?

A second problem regarding the relationship of proper nouns relates to the algorithms used for community detection. It was uncertain if communities existed in the text of the diary that, if found, might correlate with the number of intra-community relationships. Network science libraries, including those in Python, have several metrics for measuring the accuracy of communities discovered by using the number of relationships between each of these terms. This problem concerns which, if any, community detection algorithm receives a metric indicating correlation with the relationships it grouped into each community. Of the three different community detection algorithms compared for this research, the Louvain and Leiden methods were found to have a significant correlation with these relationships, while the Fluid

method was found to have an insignificant correlation with these relationships. The discovered communities were found to have significant overlap with diary entries that, in the context of the full text, sometimes refer to them directly.

A final problem posited in undertaking this research pertained to the sentiment between both the communities of proper nouns and the individual proper nouns. In some instances, it was not well-known whether significant average sentiment and change in sentiment would be seen between nations at war. The answer to this question was measured by aggregating the average sentiment across co-occurring terms while also categorizing each term as a person, nationality, product, or organization. The findings of this research indicate that such political shifts are not only detectable but measurable in sentiment over time. Additionally, the characteristics of certain nationalities closely aligned with the author's explicitly stated sense of identity. A sense of identity, as will later be explained, is a crucial aspect to identify when trying to understand how an author relates to other terms in a network.

III.Why Network Analysis Matters

Formal network analysis is necessary to analyze texts such as Anne Frank's diary since it reduces room for relative interpretation of Anne's experiences during a critical time in human history. Network visualizations and metrics are vital since they can take essential writings in history and reduce ambiguity. For example, the prejudices that were facing the Frank family may not only repeat themselves politically but may retain autobiographical signatures of someone experiencing prejudices on social media today. Refining network analysis tools, in

many ways, serves to correlate metrics and sentiment output with what the author truly felt at the time the work was created.

Modern network analysis and network science are a continuation of previous World War II research using network theory. Research by Cranmer, Desmarais, & Kirkland (2012) found that the closure of triads often could be used to predict alliances between world powers (p. 35). Moreover, their research concludes with suitable historical findings suggesting how to predict alliances in the future. By focusing on World War II, decolonization, & the post-Soviet era, Cranmer et al. (2012) were able to identify specific periods of instability that correlate with history without the need for context. This research indicates that network theory shows promise as a tool to tightly integrate the lessons of human history with the warnings that may precede undesirable recurrences in the future.

The metrics at the end of this project will illustrate the primary concerns of Anne during her time in hiding, what concepts and people mattered to her most. Individual graphs will show the associations of people critical to Anne's survival during the war, while others, such as those of the Opektra works and, in particular, Anne's relationship with Peter, will seem more representative of a typical 13-year-old girl's diary. In some cases, further research will be suggested to see if the account of this autobiographer is typical of others witnessing discrimination. Anne’s views on the negative sentiment between Christian & Jew nodes are not unlike detecting co-occurrence of other NLP use-cases on social media where enough negative sentiment should be, primarily if geographically centered, cause for humanitarian aid.

IV.Literature Review

Barnes & Harary (1983), two social anthropologists, commented in *Graph Theory in Network Analysis*, "…it seems to us as if network analysts are all too often still at the stage of children learning the concepts of arithmetic by playing with Cuisinier rods; mathematics proper still lies ahead." At the time, this was attributed to the inaccessibility of computational resources, "Empirical or imagined data are coded into graph theoretic categories, are fed into computers making use of specially written programmes, and generate useful results that in practice could not have been obtained in any other way." (Barnes & Harary, 1983). However, social networks and their measures of centrality were already being heavily researched by sociologists (Wasserman & Faust, 1994). Freeman (2004) had begun to standardize the various attempts, from earlier decades, used to describe Indian social communities, technological innovation, and urban development, "Ideally, measures should grow out of advanced theoretical efforts; they should be defined in the context of explicit process models." Freeman continued to develop these concepts into the measures of betweenness and closeness centrality that network scientists use today.

By the late 1980s, a great deal had changed with the advent of the personal computer. Developments were finally being made in the field of mathematics to unify aspects of sociology with graph theory. Researchers Love & Sloman (1995) had been independently studying what would later become PageRank centrality via their paper *Mutability and the Determinants of Conceptual Transformability*. This research would become a centrality metric used by Google Search to rank web pages. Social dynamics were now being realized in research papers more as computable attributes rather than qualitative factors.

Two decades ago, Barabási et al. (2000) discovered what is now known as a scale-free network, marking what many consider to be the beginning of network science. In *Scale-free characteristics of random networks: the topology of the world-wide web*, a breakthrough was made in understanding the self-organization of the web. These researchers learned that, rather than being random & unpredictable as previously thought, the number of vertices versus nodes followed a power-law distribution. This landmark paper not just identified the behavior but also explained the preferential attachment over time that lead to its distribution.

More recently, Blondel et al. (2008) put forward a new algorithm to identify community structure in *Fast Unfolding of Communities in Large Networks*. A significant aspect of this algorithm is that its complexity only increases linearly with the size of the network. Like that of the discovery of scale-free networks, this realization allowed massive networks like the world-wide web to be better understood with relatively little computational requirements. This algorithm, later known as the Louvain method, works in two phases that are repeated iteratively. An illustration from their research is shown in Figure A1.

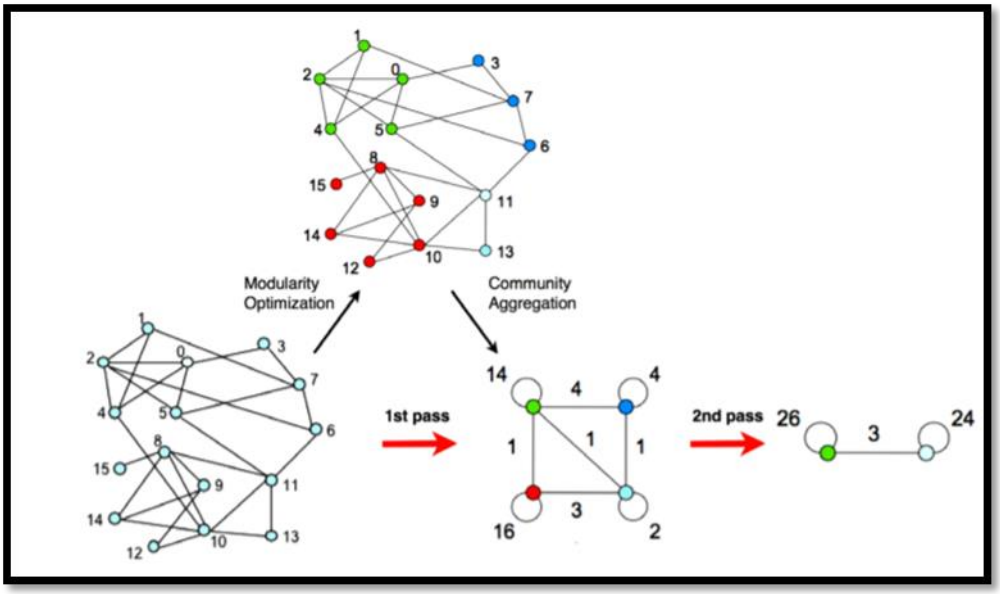

Figure A1: Community Detection via the Louvain Method (Blondel et al., 2008)

In the first phase, a metric called modularity is maximized by assigning a community to each node and seeing if intra-community edges increase by moving each node into the neighboring community. If several options increase modularity, the node is moved into the community that maximizes modularity, thereby maximizing these intra-community edges for the overall graph. The second phase is to treat each prior community as a node, connecting these new nodes with the sum of the edges that connected the communities, and repeating the first phase on this new network.

The detection of these Louvain communities has been used in recent sentiment analysis to detect extremist behavior in social networks such as Twitter. Benigni (2017) identified overt ISIS supporters using Louvain community partitioning. These Louvain communities are shown in Figure A2. Benigni aimed their research in the identification of a "passive support structure essential to the distribution of extremist propaganda." By using the research of Blondel et al. (2008), strides were being made through modularity maximization techniques of data widely available via the Twitter API. The tools that had established network analysis as a relatively new science were being used a decade later to identify forms of human behavior accurately. Furthermore, the identification of communities could be acted upon for early forms of humanitarian intervention.

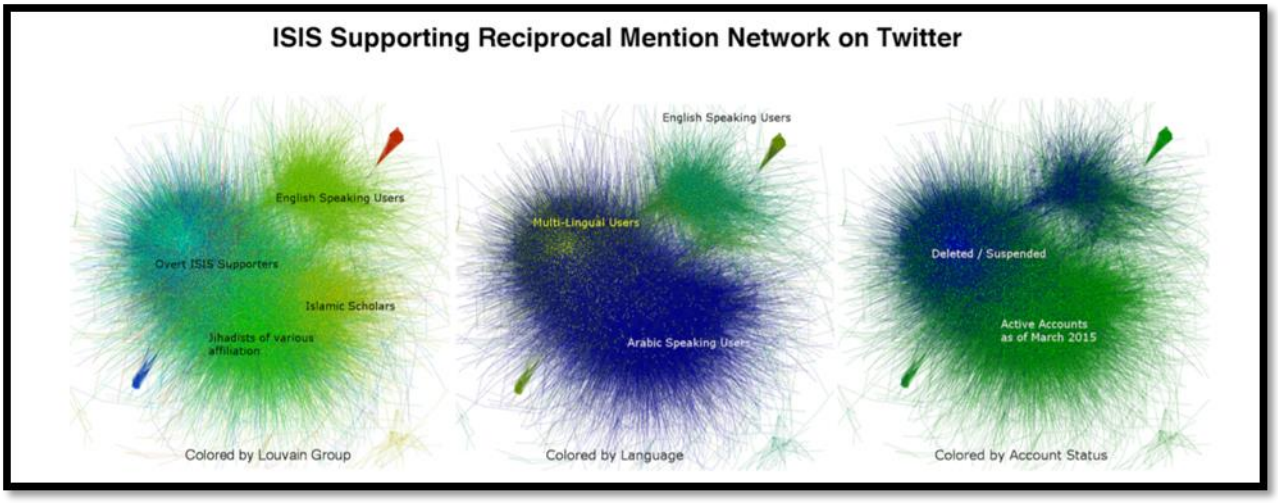

Figure A2: Community Detection via the Louvain Method (Benigni, 2017)

A great deal of exciting research had also turned to understanding similarities between literature and the relationship of characters to real-world networks. Waumans, Nicodème, & Bersini (2015) found that some of the most famous literary texts, even with their relatively small number of characters compared to the world-wide web, matched the power-law distribution of vertices like that of a real-world network closely. Figure A3 shows a degree distribution for a series in close alignment with the power-law. This high coefficient of determination value (R^2=0.8047) showed that network evolution, as seen in real-world networks, is also possible in small networks with only a single author.

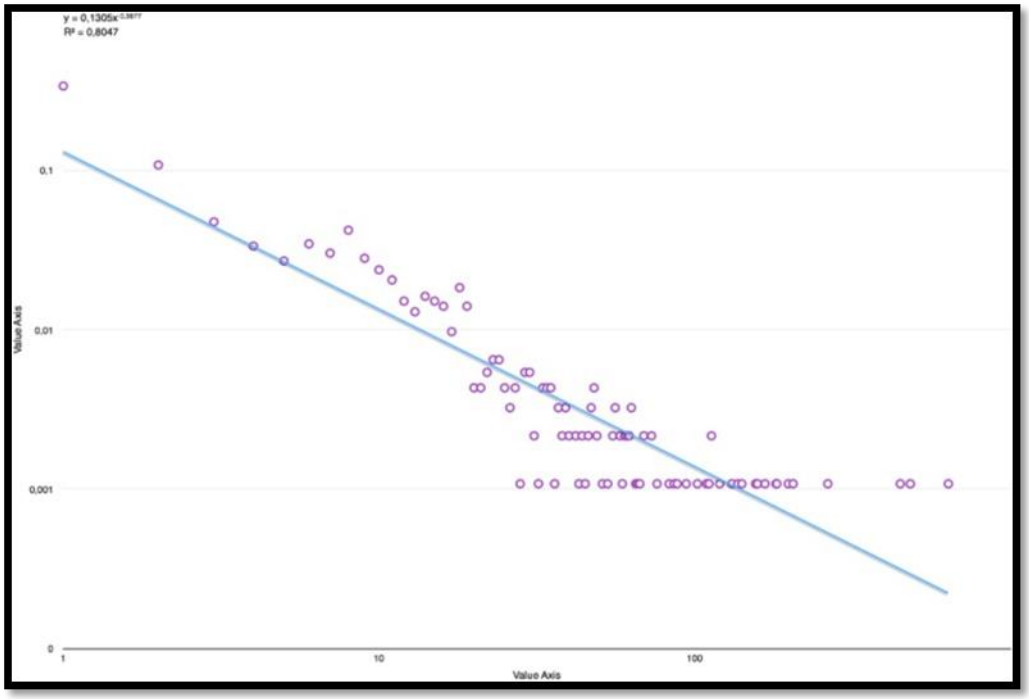

Figure A3: Degree Distribution of Harry Potter Series (Waumans et al., 2015)

Min & Park (2019) also explored character dynamics in literature, including how sentiment is transferred over time. *Modeling narrative structure and dynamics with networks, sentiment analysis, and topic modeling* covers the sentiment analysis and exchange of topics between characters written during The French Revolution. Figure A4 depicts the temporal relationship between two primary characters in this work. How characters interact and exchange information during periods of wartime were pertinent to this research, notably the communities that result from such exchanges. Min & Park's research showed evidence that narrative structure, sentiment, & the communities therein could be visualized to show dynamics that may not have been apparent to the reader.

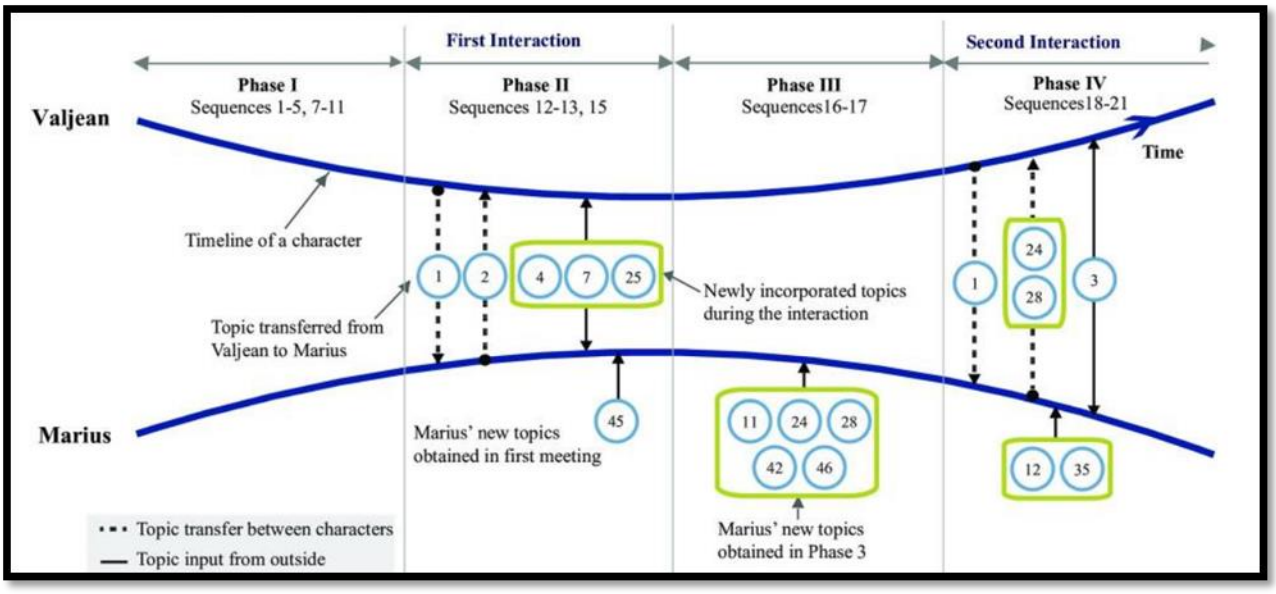

Figure A4: Temporal Topic Transfer in Les Misérables (Min & Park, 2019)

The findings of previous research in network analysis showed that it is an adept tool for extracting community partitions from text & literature as a possible form of early humanitarian intervention. Specifically, the development of characters temporally with the exchange of information showed an opportunity to extract meaning from changes in sentiment. These changes between different communities & members of those communities would serve as the foundational method in identifying what contextual overlap exists between network science & the entries of the diary. The Louvain method was used to identify communities due to its historically superior performance in maximizing intra-community edges. For comparison purposes, the co-occurring proper nouns of this text were fed into other community partitioning algorithms. Centrality metrics were analyzed to understand how each term played a role in the overall exchange of information. Finally, network statistics were used to determine if the overall graph was scale-free.

V.Historical Context

Anne Frank was born in Frankfurt, Germany, on June 12, 1929, to parents Otto (aka Pirn) and Edith while Jewish persecution was growing. Shortly after the first concentration camp was opened in 1933, Anne's family fled to Holland to escape the growing anti-Jewish sentiment. On May 10th, 1940, Germany invaded the Netherlands. Two years later, Anne received her diary just months before her older sister Margot was conscripted to serve in a labor camp. The next day, July 5th, 1942, the Frank family fled to a secret Annex above Otto Frank's offices at Opekta, a pectin and spice company, hidden by a bookcase that opened. Figure B1 shows the Anne Frank House from the front, while Figure B2 provides a cross-section revealing the secret Annex.

Figure B1: Anne Frank House (Center), Netherlands (Blazer, 1954)

Figure B2: The Secret Annex (van Wessel & Ruys, 2013)

The Frank family was aided by specific employees of the Opekta company who risked their lives to keep members of the Annex safe. These helpers included: Victor Kugler, who was then Opekta's owner since Jews could not own businesses, Miep Gies, who had been Otto's assistant, Johannes Kleiman, who was a worker at Opekta, and Bep Voskuijl, who had worked for Otto. One week after going into hiding, the Frank family was joined by the van Pels family (pseudonym van Daan), who had also gone into hiding after being summoned for forced labor in the concentration camps. The van Pels family consisted of Peter, 15, who was the only child of Auguste (pseudonym Petronella) and Hermann (aka Putti). Peter and Anne had a brief but meaningful relationship carefully documented in Anne's diary. The final addition to the Annex was dentist Fritz Pfeffer (pseudonym Albert Dussel) on November 16th, 1942.

On August 4th, 1944, an anonymous tip-off resulted in a raid by police whereby all in hiding, as well as the two men assisting the families, were arrested. The Jewish families tragically did not survive the concentration camps except for Otto Frank, who went on to publicize the contents of Anne's diary given to him by Miep. Johannes was released from Amersfoort camp after just one week due to being in ill-health and returned to run Opekta. Victor escaped en route to a labor camp during an Allied bombing and returned to business in Amsterdam, and later in Canada. Today, the Anne Frank house has been restored as a biographical museum devoted to telling Anne's story as well as educating the public about forms of discrimination.

VI.Data & Network Analysis

a.NLP Tags

Figure C1 shows the Natural Language Processing (NLP) tokenization results from a sample sentence taken from the diary during part-of-speech (POS) tagging. Proper nouns 'Bep', 'Margot', and 'Bijenkorf' have been differentiated from the nouns 'me' and 'skirts.' The English word 'skirt' could have also been detected as a verb. Misidentification was avoided due to the overall relationships constructed by NLP (Manning & Schutze, 2008). Part of this relationship is the verb 'bought' having associations to the proper noun using it as well as the subsequent adpositions and direct object 'skirt.' Since this direct object is being acted upon by another verb, NLP tags this entity accordingly (Honnibal & Johnson, 2015).

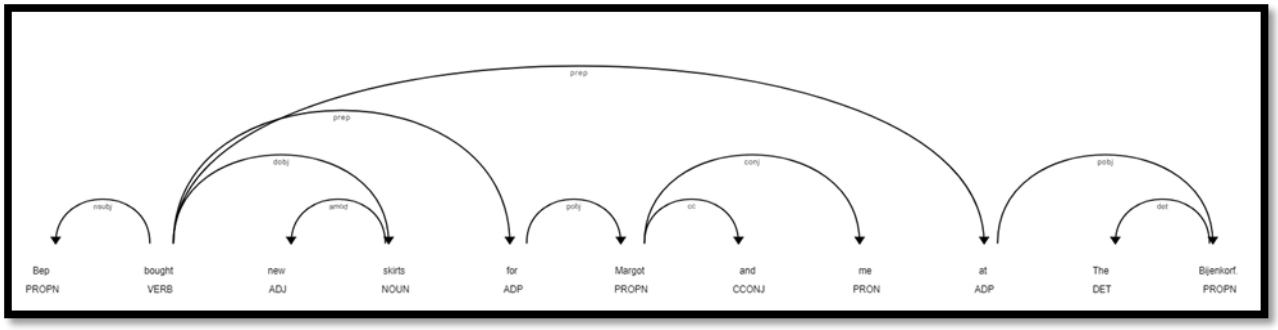

Figure C1: NLP Relationships

Named entity recognition is a feature of the spaCy library used to obtain a real-world object association for each noun. spaCy uses a statistical model strongly dependent on the examples on which they were trained. Several differently sized models are offered by spaCy based upon how standard the use of the English language is for that use case. Smaller training sets offer better performance for use cases involving standard text. Since the highest degree of accuracy was desired and performance was not an issue, en_core_web_lg, the model from the most extensive training set, was chosen. The en_core_web_lg library is based on two data sources: Common Crawl and OntoNotes. Common Crawl is an open-source web crawl dataset in 40+ languages, including German. OntoNotes is a more focused open-source dataset containing text from news, broadcast, talk shows, & weblogs. Together, they provided a varied training set that would come to be essential in analyzing the unedited diary of Anne Frank.

Sentential co-occurrence required the use of spaCy's feature for sentence boundary detection (SBD). SBD correctly identified sentences from the full text despite quoted inner-sentences, returns that occurred mid-sentence, and abbreviations with periods. A dictionary was made of all proper nouns detected along with their frequency of occurrence in the overall diary. From this dictionary, the few typo's and other mistakenly detected proper nouns were

removed. This dictionary was first used as a model for the diary co-occurrence of unique terms. The list of sentences generated from SBD was then checked for the sentential co-occurrence of unique terms from the prior dictionary. In this way, a new dictionary was created out of tuples representing co-occurrence & the sentences they occurred in being their values. The library natural language toolkit (NLTK) was used in Python in this new dictionary to assess overall sentiment in each sentence, and the new dictionary was updated with average sentiment for each tuple (Liu, 2015; Bird, Klein, & Loper, 2009).

For an additional look at the overall diary before diving into co-occurrence, Figure C2 displays a word cloud of the proper nouns for all tag types. As one might expect, the inhabitants of the Annex occupy the most dominant terms in this figure. What also comes to light are words like Jewish, German, British, Dutch. These words fall into the NORP (nationalities or religious or political groups) NLP tag. This categorization will be of interest later in this project when co-occurrent terms are clustered by the NLP tag type. Since it is a surname in Dutch, The word “van” is one of the largest since it encompasses the van Daan family, Mr. van Daan, Mrs. van Daan, and other occurrences of less popular characters outside the Annex. For this reason, further analysis in co-occurrence seeks the fullest proper noun if it is present in a sentence. For example, the names van, D., van D., and Mr. van D. will become Mr. van D if it is available in the sentence. Despite this, family names (van D.) were often more frequent than their counterparts.

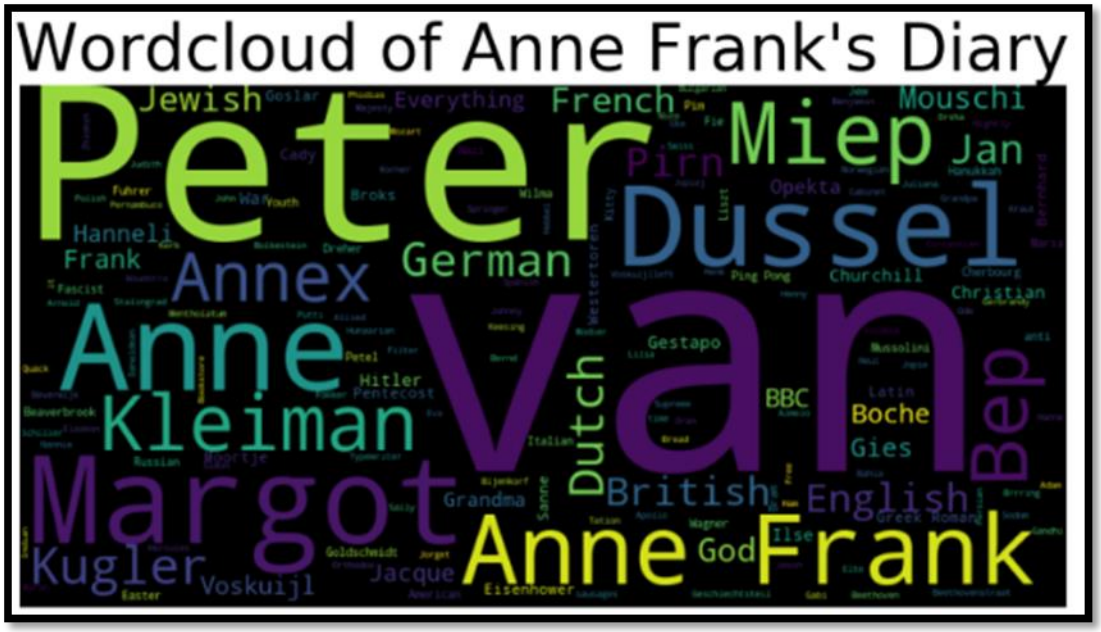

Figure C2: Wordcloud of Anne Frank's Diary

If not engineered correctly, it is essential to note that any automated text analysis fed into network libraries will lead to highly inaccurate results. One drawback to analyzing historical text such as a diary is that they are prone to typos as well as expressions/words in other languages. This difficulty is a hurdle when aggregating nouns since unknowns can easily be detected by ML-based libraries as a proper noun. Words like "entsetzlich" (German for terrible) had to be manually removed from the list of detected proper nouns.

b.Time Series

Figure D below shows how the different NLP tag counts changed throughout the diary entries. The Person tag is consistently the most prevalent NLP entity, whereas Product is the most uncommon. Specific periods, such as the invasion of Sicily on July 10th, 1943 to September 10, 1943, when Anne gets news of Italy's surrender, are marked by the increased mention of

overall tag count. The increased variance in all tags towards the final entries in the diary is also noticeable in Figure D.

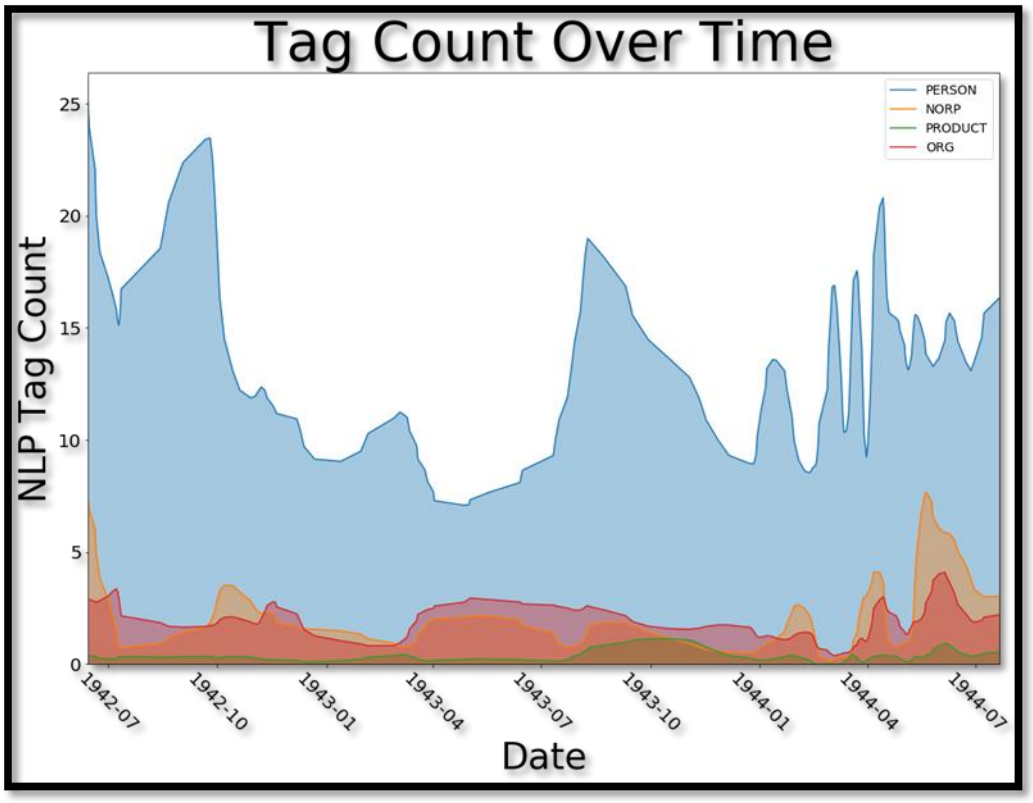

Figure D: NLP Entity Types by Date

The most significant drop-off of tag occurrence is following the July to October period in 1942 when the Frank family goes into hiding. During these four months, living in such close quarters, the families in the Annex clashed frequently. Anne is particularly prone to telling full accounts of the rituals of each family in this period, including the reprimanding of Peter. Immediately after these four months, her diary entries turn to shorter anecdotes reflecting on her father's brief illness, events surrounding the Annex, and even excitement for the addition of Albert Dussel to the group in hiding. With the addition of someone to their relatively small quarters, the excitement of this change was short-lived.

Figure D's tag count, at any given date, is not able to distinguish between overall excitement/animosity that may be occurring between members of the Annex versus proper nouns affecting the events surrounding them. In this way, the NLP tag count cannot in-and-of-itself be used to detect if a dramatic turn of events is unfolding in the circumstances surrounding the author. The period of relatively high variance could also have indicated duress of outside influences on the Annex. That possibility, too, is impossible to separate from interpersonal events. This period is marked both by Anne's struggle to have her father accept her relationship with Peter while also carefully documenting events such as an assassination attempt on Hitler. Measuring attributes about the occurrence of NLP tags is not enough to differentiate significant historical shifts from interpersonal ones. Separating the occurrence count of German & British also would not provide an indicator since proper nouns can be used as adjectives when describing interpersonal relations. Attributes about co-occurrence had to be investigated in order to gain further insight into the relationships between proper nouns.

c.Networks of Co-Occurrences

Next, the overall graph was visualized for all terms shown in Figure E1 below. Here, node size represents the frequency of occurrence for each term in all diary entries, and edge width represents the co-occurrence frequency of the connected terms within diary entries. Like the word cloud, specific terms occur far more frequently than others. It can also be observed that specific terms in Figure E1 reside in individual clusters of more densely blue (higher degree) regions to other nodes in their respective cluster. One notable cluster, in the bottom right of

Figure E1, contains nodes well-connected to each other but less connected to everything else. NLP as a community as well as other community types will be investigated throughout this project to see what terms can best be grouped w/r/t different measures of network metrics.

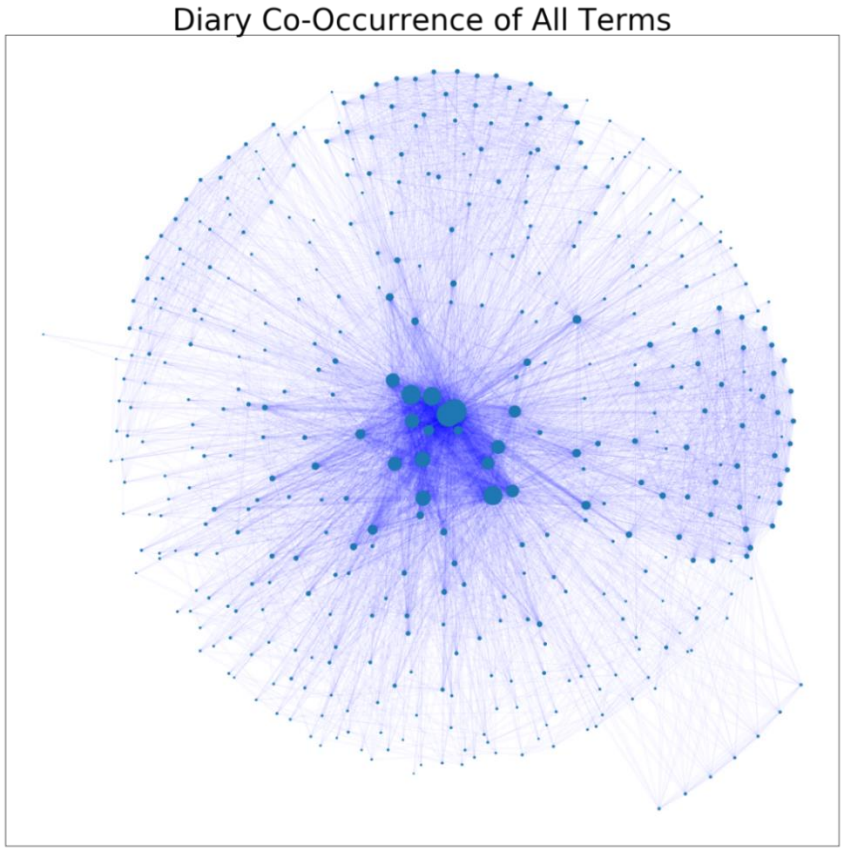

Figure E1: Overall Graph of Diary Co-Occurrence and Term Frequency

Without yet knowing the NLP type of these nodes, one might wonder why the aggregation of diary co-occurrence results in this structure. It will be investigated what types of nodes in the center of this graph relate to the outlying clusters w/r/t all the measures of centrality that exist in NetworkX. Certain cliques, or subgraphs whose members are all fully connected by vertices, will emerge from this process (Wasserman & Faust, 1994). As it pertains to the context of the diary, reasons citing quotes from the text will be offered for why specific proper nouns connect these cliques. A clique is a If and how this is closely aligned with the

narrative told by Anne Frank will be discussed as it pertains directly to the goal of establishing if network theory is a suitable tool in analyzing historical texts.

Taking a closer look at some of the most common terms in diary co-occurrence, Figure E2 shows a similar representation but now with some of the most common proper nouns labeled. Due to the disparate frequency of co-occurrence, the edge width was taken as the log of the co-occurrent frequency while node size remains term frequency. Kitty, Anne's pet to whom she addressed almost all her entries, appears as the most frequent term along with other members of the Annex and the NORP's affecting the events surrounding the Annex. Edge connections between Jew & Hitler and Jewish & Margot tell a fuller story about the events detailed in Anne's diary.

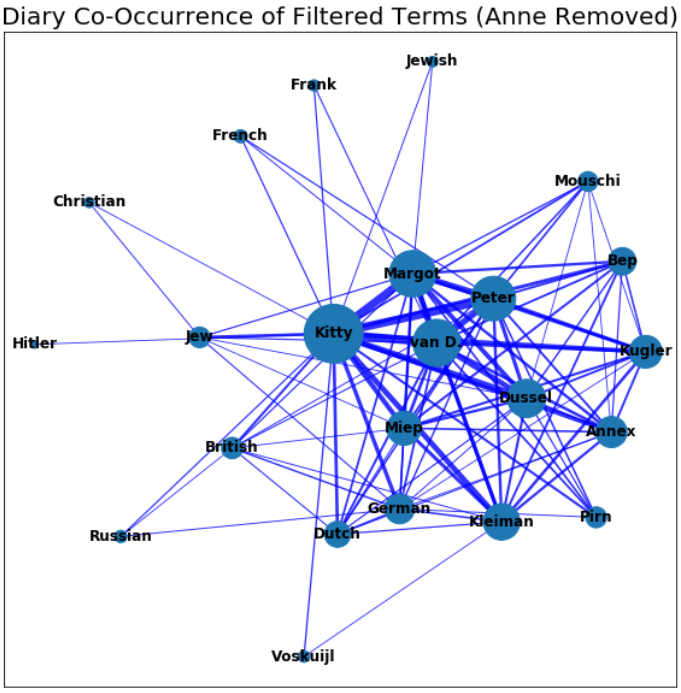

Figure E2: Frequent Terms of Diary Co-Occurrence

d.Sentiment Analysis

To more clearly see how these terms relate, sentential co-occurrence was next used to specify edge connections, as shown in Figure F1 below. The color of edges represents the average sentiment when the connected terms co-occurred in sentences throughout the diary. The sentiment between Jew & Christian is, on average, negative, as is the sentiment when Jew & German co-occur in sentences. The same is true between many NORP terms, though, it is hard to observe without clustering by NLP attribute. The frequency of Jew & Christian co-occurrence can also be seen to be larger than most other edges involving NORP's, comparable to the frequency of co-occurrence between Persons.

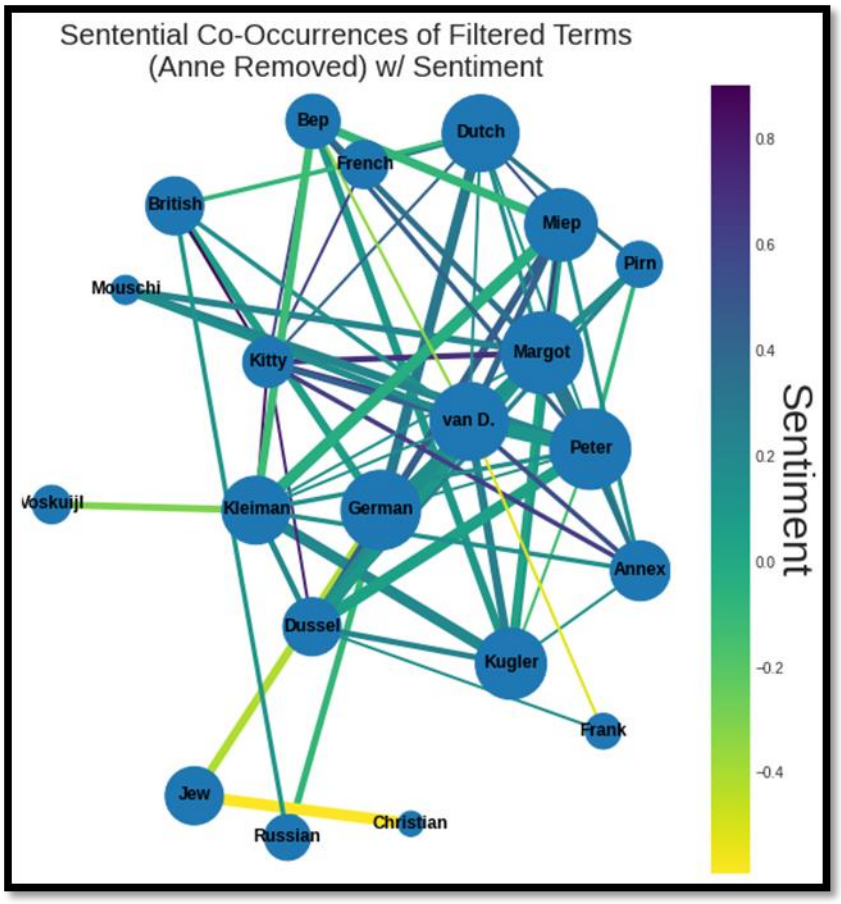

Figure F1: Sentiment of Sentential Co-Occurrence for Frequent Terms

As was seen in the previous graphs, certain regions of well-connected terms with high sentential co-occurrence emerge even after less common terms have been filtered off. Kitty, Peter, Annex, and other terms central to these graphs are surrounded mainly by positive sentiment. To see the weighted sentiment surrounding each node, Figure F2 shows the average sentiment related to each term.

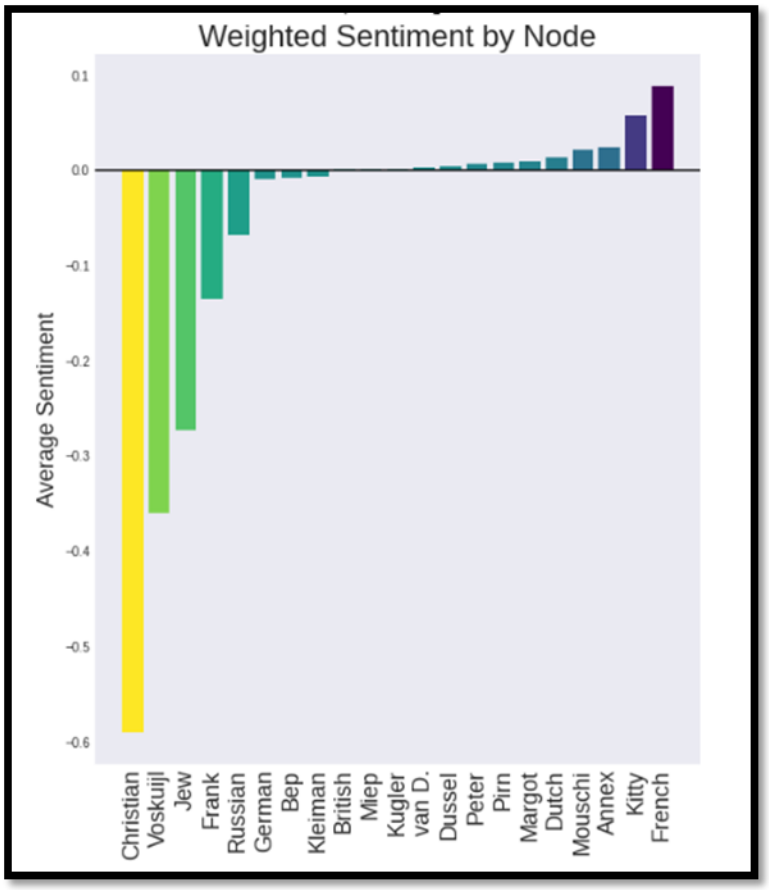

Figure F2: Weighted Sentiment by Node

The term Christian only had one edge with strongly negative sentiment and remains the lowest bound of sentiment for this graph, followed shortly by Jew due to their relationship. The term Jew, however, has less negative sentiment due to its relationship with Dussel. Voskuijl, Bep's last name and the name of her husband, has the second-lowest weighted sentential co-

occurrent sentiment. This finding was due in large part to Anne's dismay over Mr. Voskuijl's cancer diagnosis in June 1943. The maximum negative sentiment (-0.6) far exceeds the maximum positive sentiment (0.1), whereas more of these frequently occurring proper nouns have an overall positive sentiment rather than negative. This finding initially seemed to indicate that Anne might have been focusing on positive concepts with her characters instead of focusing on her dire situation. This theory was previously suggested when viewing the node size of characters inside the Annex (and positive sentiment around them), which were often more substantial than the node sizes of NORP's that frequently discuss the situations surrounding the Annex. For things Anne disliked, it was also suspected that Anne felt more impassioned with negativity about those terms than the sentiment about things she viewed positively. Both assumptions were tested for correlation later in this research.

Not all NORP's sententially co-occurred, on average, with negative sentiment. Dutch and French appear with positive sentiment, French even exceeding the average positive sentiment of all terms. Anne's lessons in French were a source of excitement for her early on in her existence in the Annex as well as later during lessons in which she visited Peter to teach him French. The term Dutch is a NORP that pervaded the Annex with Dutch broadcasts, Dutch supplies, and, more generally speaking, Anne's love of Dutch culture, "I love the Dutch, I love this country, I love the language, and I want to work here. And even if I have to write to the Queen herself, I won't give up until I've reached my goal!" Another aspect of this is the fact that the Dutch city of Rotterdam had been early-on bombed. This bombing was a surprise escalation tactic employed by the Germans to force the expedited surrender of the Netherlands. The early

capture of this country contributed to less Dutch mentions by Anne of negative sentiment relative to the fighting amongst other nationalities during this period.

e.Time Evolution of Sentiment

Looking at just the sentiment in the year 1942, Figure G1 shows how the previous sentiments began during Anne's first year in hiding. Negative sentiment between the Frank and van D. family is a result of the tension these family members faced while now having to live in close quarters. The exception to this was Anne's growing feelings towards Peter. Anne also grew to appreciate how Bep would provide a much-needed relief from outside the Annex with gifts, "We have a nice treat in store: Bep's ordered a correspondence course in shorthand for Margot, Peter and me." The relationship is reflected as positive sentiment between Peter & Bep. Dutch is the only NORP that makes an appearance as a common term in sentential co-occurrence.

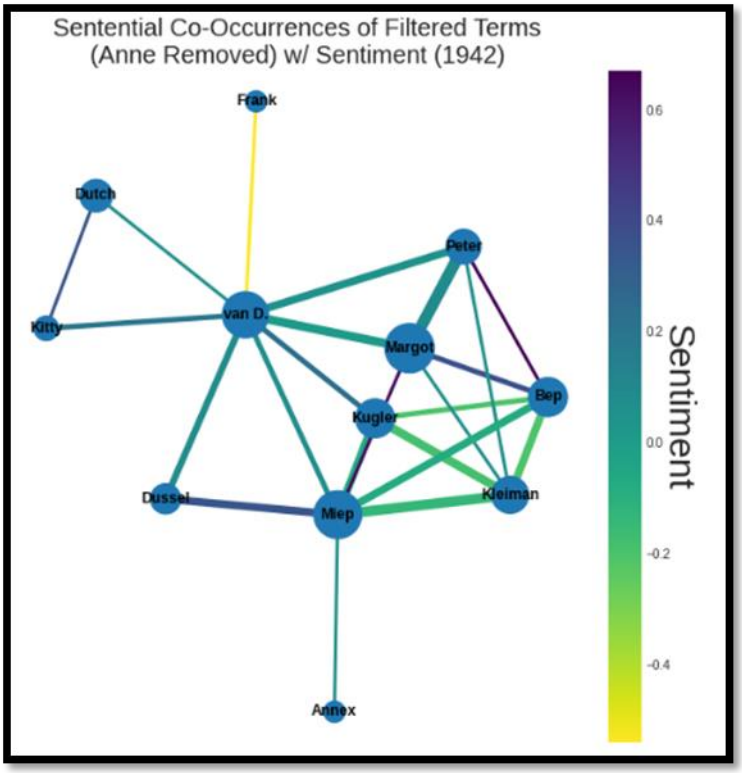

Figure G1: Sentiment of Sentential Co-Occurrence for Frequent Terms in 1942

In 1942, the sentiment between individual members of the Annex was more negative than it was for the overall diary, most notably the sentiment between the four helpers of the Annex: Bep, Kleiman, Miep, & Kugler. Anne was often concerned about the burden her existence put on them, “We always thought there was enough room and food for one more person, but we were afraid of placing an even greater burden on Mr. Kugler and Mr. Kleiman.” Over the next two years, this negative sentiment subsided as the arrangement the members of the Annex had with the helpers became the new normal.

Figure G2 shows the sentiment in sentential co-occurrence for the year 1943. Frank is no longer a frequently co-occurring term, and the van D. family is surrounded by mostly positive sentiment. The exception here is the negative sentiment between Putti and van D. Putti was Mrs. van Daan’s pet name for her husband, Hermann. Mr. van Daan was often

exhausted by Mrs. van Daan's fears, including her fear of burglars in the night, and this was a source of much amusement for Anne. Sentiment around the Annex helpers this year is generally better except for Bep, who is surrounded with strongly negative sentiment, "Bep had a nervous fit last week because she had so many errands to do. Ten times a day people were sending her out for something, each time insisting she go right away or go again or that she'd done it all wrong." As members of the Annex grew accustomed to their new lives, it seems as though Bep's noted willingness to help had been exploited to the point of exhaustion. Anne spent less time noting her fear of the helpers being exposed to NORP's and more time reflecting on the effect these chores had on those helpers.

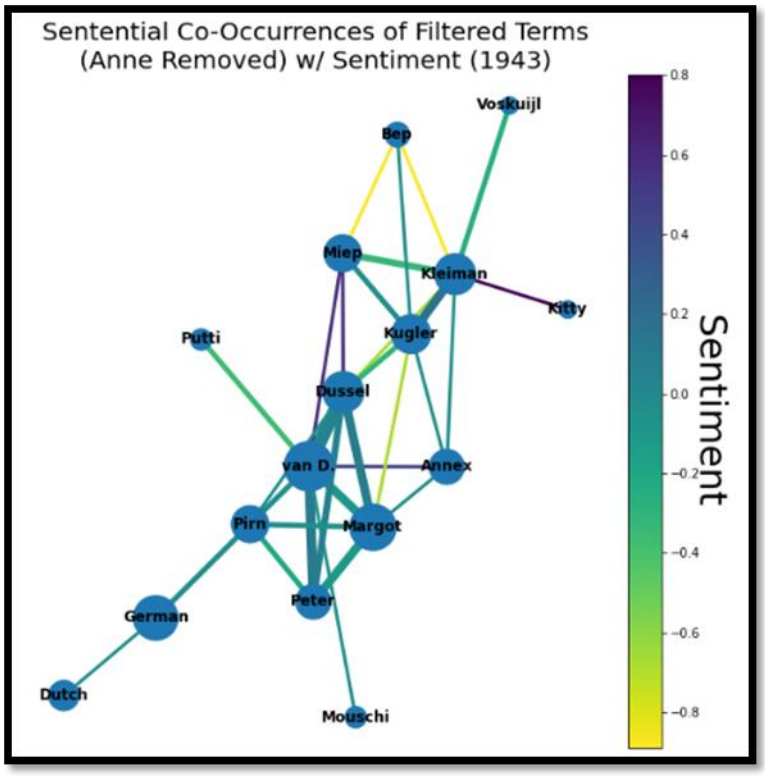

Figure G2: Sentiment of Sentential Co-Occurrence for Frequent Terms in 1943

Anne's final year in hiding exhibited very positive sentiment between the members of the Annex, as shown in Figure G3. NORP's showed neutral to negative sentiment, more

negative specifically between the terms British, German, Russian, & Dutch. The term French has a more neutral sentiment due to the positive sentiment from her French lessons averaged with negative sentiment from the war. Interconnectivity can be seen between the NORP's that differ from the members of the Annex. As was explored later in the project, this separation of interconnectivity allows different edge density analyses to discover that NORP's often behaved as a separate community. However, it can already be seen without community clustering in Figure G3.

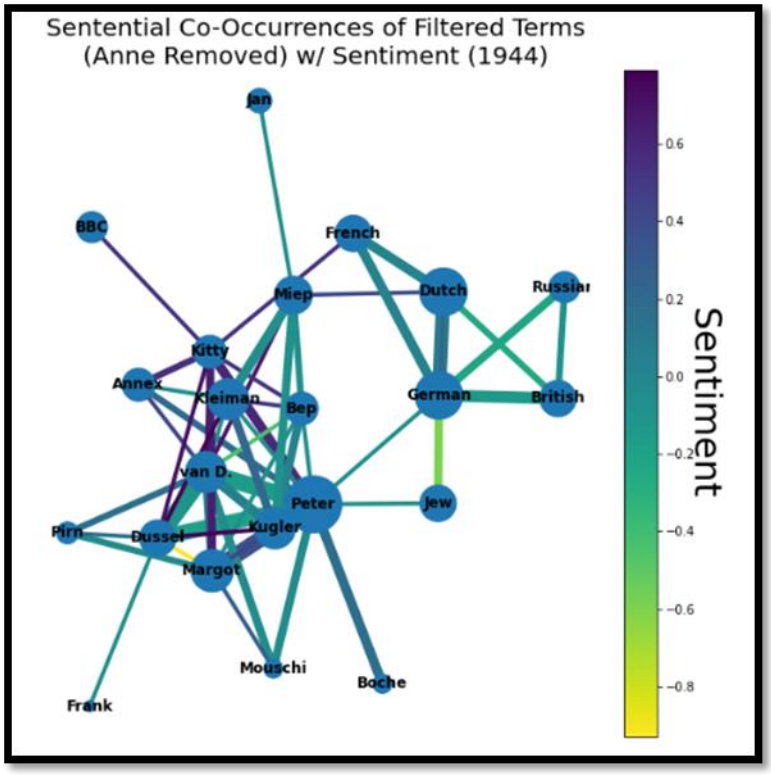

Figure G3: Sentiment of Sentential Co-Occurrence for Frequent Terms in 1943

In order to incorporate the element of sentiment over time, sentential co-occurrence was reimagined in Figure G4 with edge color representing the slope of sentiment over the days of each diary entry. German & British had a strongly positive slope as the diary unfolded, whereas British & Dutch had a noticeably negative change in sentiment. This finding was

surprising since when either of these pairs co-occurred, it always interested Anne for the same reason: the liberation of Holland. However, as the diary progresses, Anne gradually loses faith in the British, “England must fight, struggle and sacrifice its sons to liberate Holland and the other occupied countries. After that the British shouldn't remain in Holland: they should offer their most abject apologies to all the occupied countries, restore the Dutch East Indies to its rightful owner and then return, weakened and impoverished, to England. What a bunch of idiots.” Taking a closer look at sentences where German & British co-occur, early entries in the diary are strongly negative due to more accurate depictions of war, e.g., the inclusion of the word ‘firebomb.’ Anne spends less time describing warfare between these two NORP’s and more time criticizing the British. The result is German & British having a strongly positive slope throughout the diary; however, as was seen earlier in this research, the resultant average sentiment is neutral.

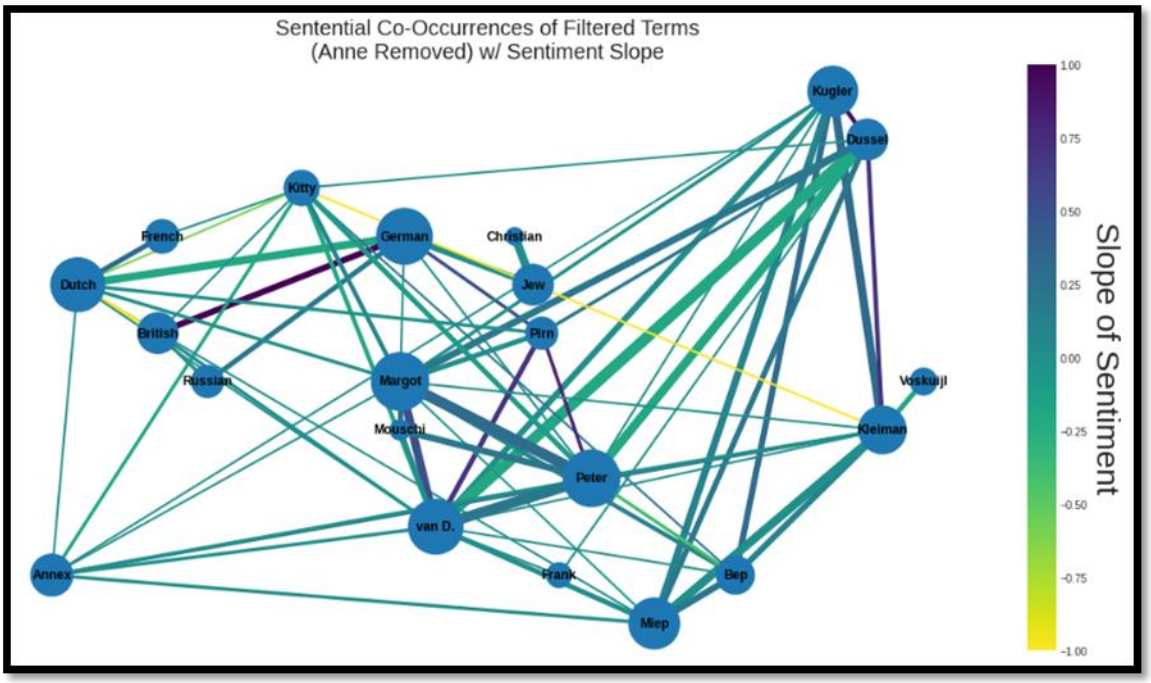

Figure G4: Sentiment Slope of Sentential Co-Occurrence for Frequent Terms

The term Peter in this figure does not have as strongly positive or negative change in sentiment as the previous terms, but every edge connecting the term to others is both wide and shows a slight positive change in sentiment. Given their relationship, we know Peter should be a term of increasing positive sentiment, and the evidence is bolstered by the higher frequency of sentential co-occurrence as reflected by edge width. It may be possible that smaller edges with less data of co-occurrence available, such as that between British & German, may be less reliable to overlap with the author's actual overall opinion. Lack of overlap from limited data is also hindered by sentiment detection libraries being vulnerable to inaccuracies from less common uses of language, e.g., sarcasm.

Figure G5 shows the change in sentiment over sentential co-occurrence for 1943 only. Python's Graphviz library has automatically organized two separate regions of mixed positive and negative change in sentiment. These two regions are separated by a region of relatively neutral sentiment. Negative change in sentiment between Dussel and the van D. family was well documented in 1943, hence the edge width between these two terms. Families were primarily excited in 1942 for Dussel, the newest addition to the Annex. This sentiment soured in 1943 when they began to argue, "At the moment, the tempestuous quarrels have subsided; only Dussel and the van Daans are still at loggerheads. When Dussel is talking about Mrs. van D., he invariably calls her 'that old bat' or 'that stupid hag,' and conversely, Mrs. van D. refers to our ever so learned gentleman as an 'old maid' or a 'touchy neurotic spinster, etc.'"

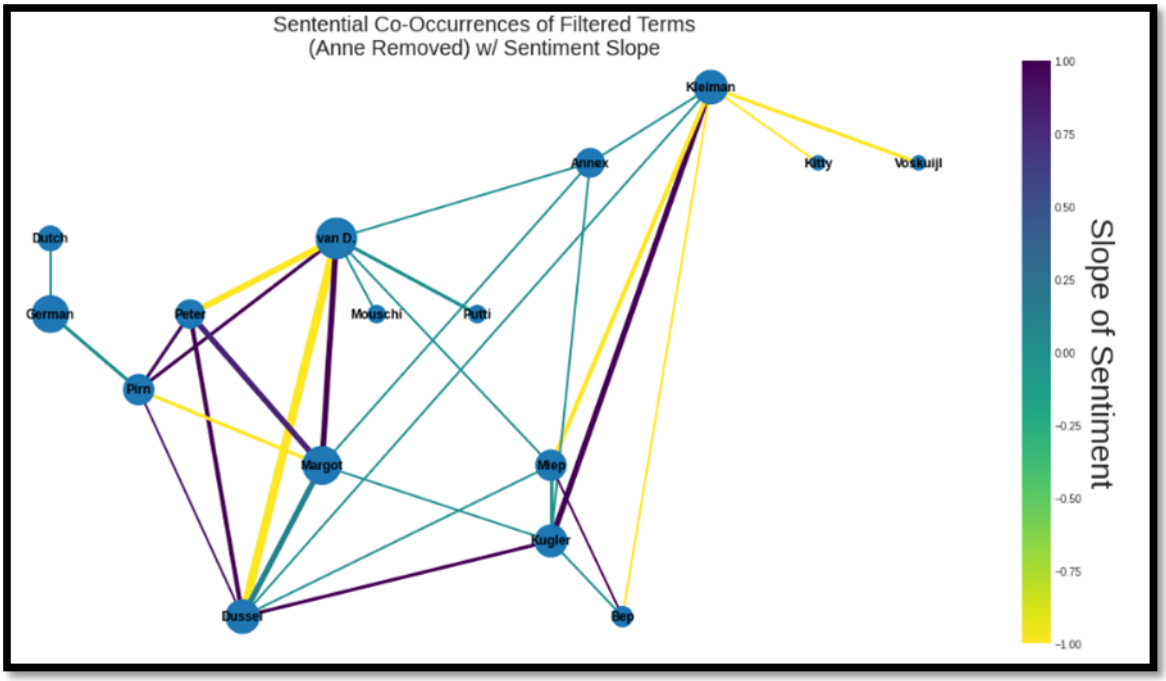

Figure G5: Sentiment Slope of Sentential Co-Occurrence for Frequent Terms (1943)

Neutral sentiment slope at the center of this graph deals primarily with the Annex. It also involves members of the Annex and helpers of the Annex. In 1943, although notable sentiment surrounded these terms w/r/t diary entries, Anne was more verbose in her writing such that they did not sententially co-occur. For example, someone reading the following entry would take away negative sentiment between a member of the Annex and a helper, “For Hanukkah, Mr. Dussel gave Mrs. van Daan and Mother a beautiful cake, which he'd asked Miep to bake. On top of all the work she has to do!” This lack of aggregation was found to be one weakness of analyzing sentential co-occurrence. If authors spend more time describing the relationship between two terms, it is possible that they may not sententially co-occur & be skipped.

f.Ego-Centric Subgraphs

By next focusing on specific nodes (ego) and only those that shared edges with that specific node (alters), the relationships between immediate neighbors were explored (Perry, Pescosolido, & Borgatti, 2018). Figure H1 shows the ego-centric graph of Kitty. Reducing the cut-off for node frequency revealed negative sentiment between Kitty and the terms Cabinet & Bolkenstein existed due to the use of the word 'war' in the following sentence, "Dearest Kitty, Mr. Bolkestein, the Cabinet Minister, speaking on the Dutch broadcast from London, said that after the war a collection would be made of diaries and letters dealing with the war." In some ways, given that this is a diary written during wartime, it might one day be prudent to try to find words to exclude from sentiment detection -or, possibly, a library that normalizes words of extreme sentiments given their frequency.

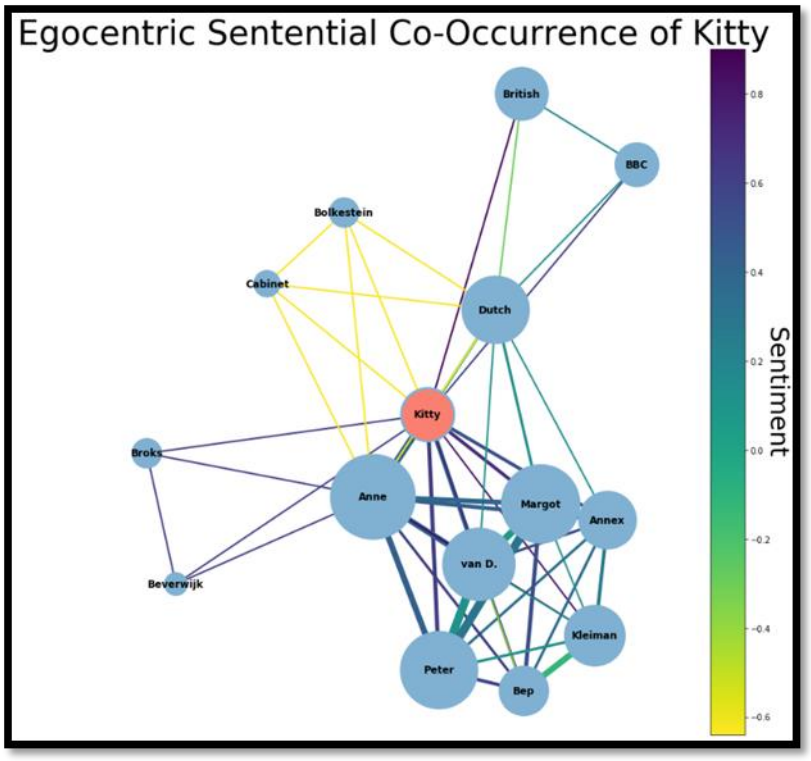

Figure H1: Sentential Co-Occurrence of Kitty's Egocentric Network

Figure H2 shows the egocentric network of Anne. As the ego, more positive sentiment surrounds Anne. Highly positive sentiment is most notable between Anne and Kleiman & Miep to whom (like all helpers) she was very grateful for, Peter to whom she had a romantic relationship, Margot, and the Annex that provided her protection. Unlike the egocentric graph of Kitty, Anne's egocentric graph has less variance in sentiment. Since the first sentence of every entry was addressed to Kitty but rarely related to her cat, it makes sense that sentiment around Kitty may have been more disparate.

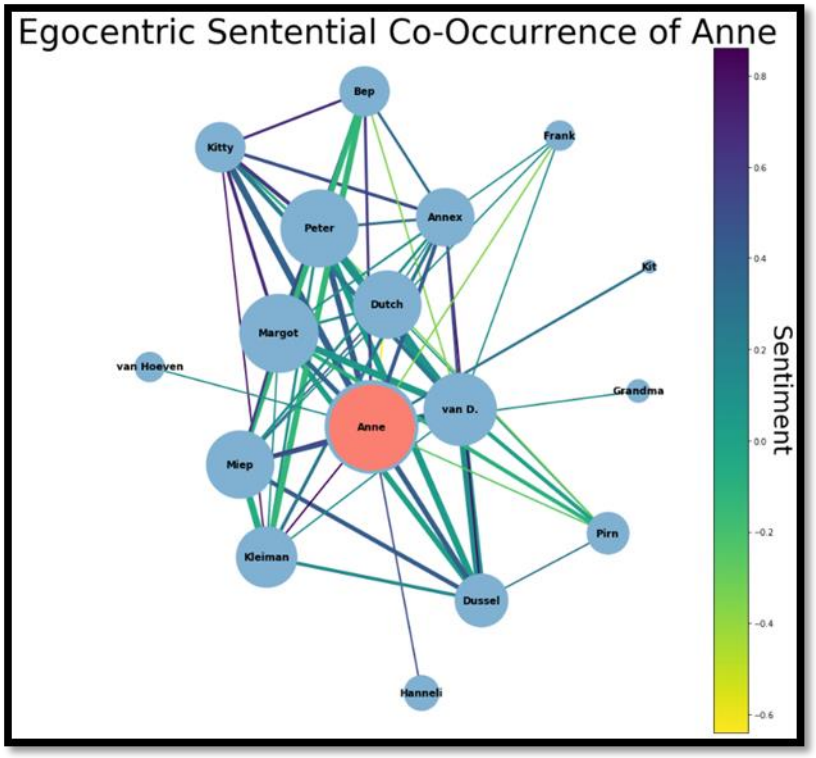

Figure H2: Sentential Co-Occurrence of Anne's Egocentric Network

Figure H3 shows the egocentric graph of Annex. Unlike the other terms, Annex has edge widths to the alters that are relatively smaller than the alters have with each other. The dissimilarity is due to Anne writing more about the occupants of the Annex than of the Annex itself. The negative sentiment between Annex & van Maaren refers to Wilhelm van Maaren

who became warehouse manager in the spring of 1943 but was unaware of the inhabitants of the Annex (though, he later grew suspicious of intruders). Anne frequently suspected Wilhelm van Maaren would betray them and posed a direct threat to the safety of the Annex. It is worth noting that historians currently do not know who betrayed the members of the Annex.

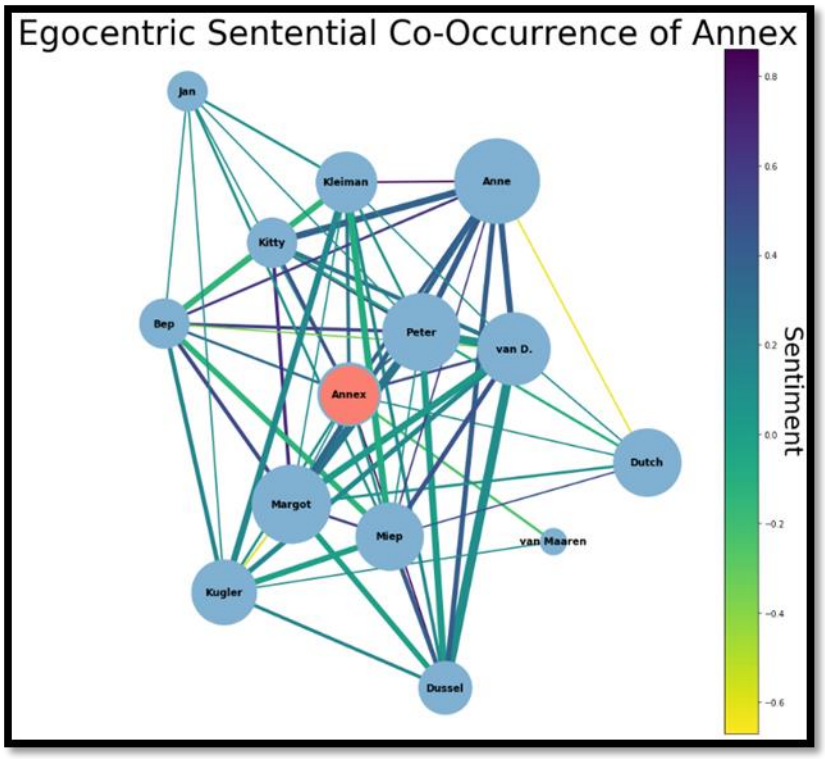

Figure H3: Sentential Co-Occurrence of the Annex's Egocentric Network

g.Community Partitions & Assortativity Analysis

In order to more clearly distinguish sentiment between NLP attribute types, frequent terms were first grouped by the NLP attribute. The average sentiment, as well as co-occurrence between them, were visualized in Figure I1. The average sentiment between NLP attributes is generally neutral to positive, most notably between Person & Product and ORG & NORP. Co-occurrence frequency of the people Anne lived with contributes to this node size as reflected by

the largest node being Person. On average, the sentential co-occurrence frequency was largest between Person & ORG, as reflected by the largest edge width. Here, Miep was detected as an ORG rather than a Person by spaCy. Miep being an ORG may, in part, be due to her diligent services to supply the Annex occupants with resources, “Miep is just like a pack mule, she fetches and carries so much. Almost every day she manages to get hold of some vegetables for us brings everything in shopping bags on her bicycle.” Annex is the node tagged as Product by spaCy since it is closest to the library’s definition of “objects, vehicles, foods, etc.” and does not fulfill a service often mentioned in the diary.

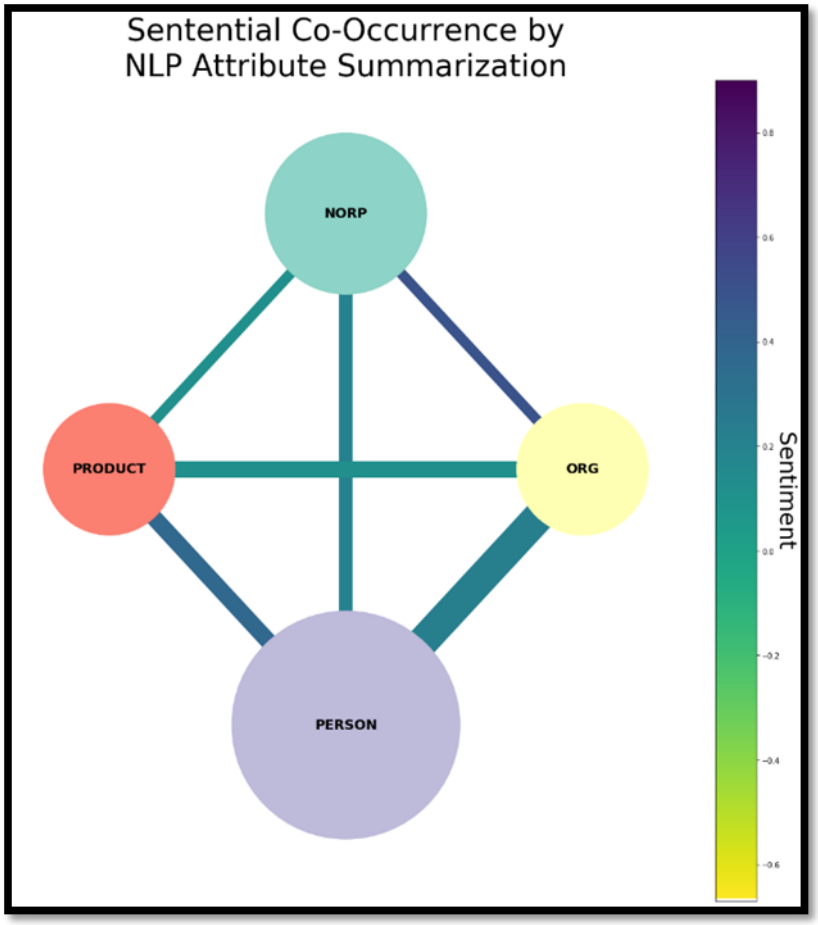

Figure I1: Sentiment of Sentential Co-Occurrence by NLP Attribute (Summarization)

A separate method of community partitioning was done using modularity maximization (Newman, 2019), as implemented by the Louvain algorithm. This method works by maximizing intra-community edges and minimizing the density of edges outside each community. Although clustering by NLP Attribute relies on a well-establish ML library in spaCy, it also relies on text-based training that may be dissimilar from Anne Frank's diary. The NLP detection of each term's attribute does not factor in the density of co-occurrence like that of the Louvain method. Figure I2 shows the Louvain communities with edge density as a factor. Figure I3 shows the membership of these communities.

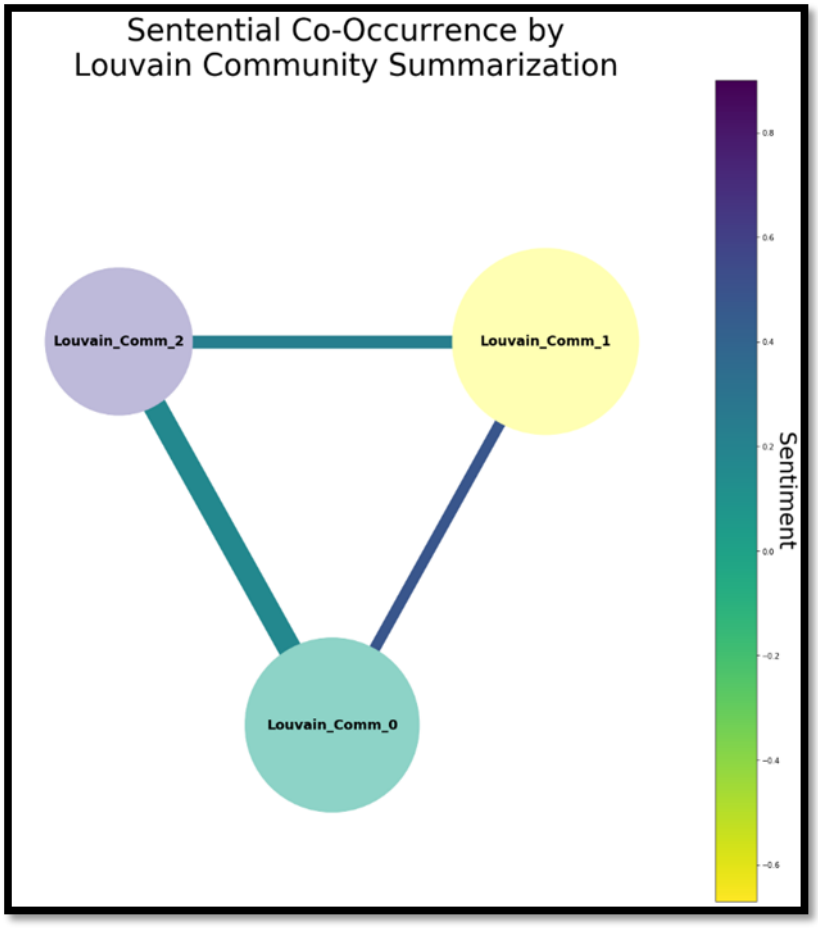

Figure I2: Sentiment of Sentential Co-Occurrence by Louvain Community (Summarization)

Figure I3: Louvain Community Membership

Louvain Community 0 contains members of the Annex aside from her family and the cat, who reside in Louvain Community 2. Louvain Community 1 shows mostly what the spaCy referred to as NORP's with the addition of Kitty. The membership of Figure I3 shows that Figure I2's positive sentiment between Louvain Community 0 & Louvain Community 1 existed between members of the Annex and NORP's. It also shows that their sentential co-occurrence, via edge width, is the lowest. The average node size in Louvain Community 1 was shown as the largest, thereby indicating these terms, on average, sententially co-occurred most frequently.

To further inspect the interplay of sentiment between nodes in clusters, Figure I4 shows the graph of frequent sentential co-occurring terms clustered by the NLP attribute. Here, more insight is given regarding the sentiment between Person & Product (the Annex) and why the summarization showed overall positive sentiment between the two. Kitty, the van D. family, and Bep can be seen to influence the sentiment with the Annex. The relationship between ORG & NORP is seen to be solely influenced by Miep's positive sentiment with Dutch, "You can see we're never far from Miep's thoughts, since she promptly noted their names and addresses in case anything should happen and we needed contacts with good Dutch people." In contrast with the Louvain method community detection approach, the nodes below are colored by Fluid community. This community detection algorithm focuses less on edge density and treats each attempted community as a fluid in a non-homogeneous environment (Parés et al., 2017). Fluid

communities had a lower assortativity versus NLP communities. Attribute assortativity (referred to here just as 'assortativity') is defined as the tendency for members of a group sharing the same attribute to associate with one another (Newman, 2019). Fluid communities can be seen to be disassortative in Figure I4, or members of that community tend to interact with others of different communities, whereas NLP communities can be seen to be assortative. Community detection method metrics were later compared as part of this research.

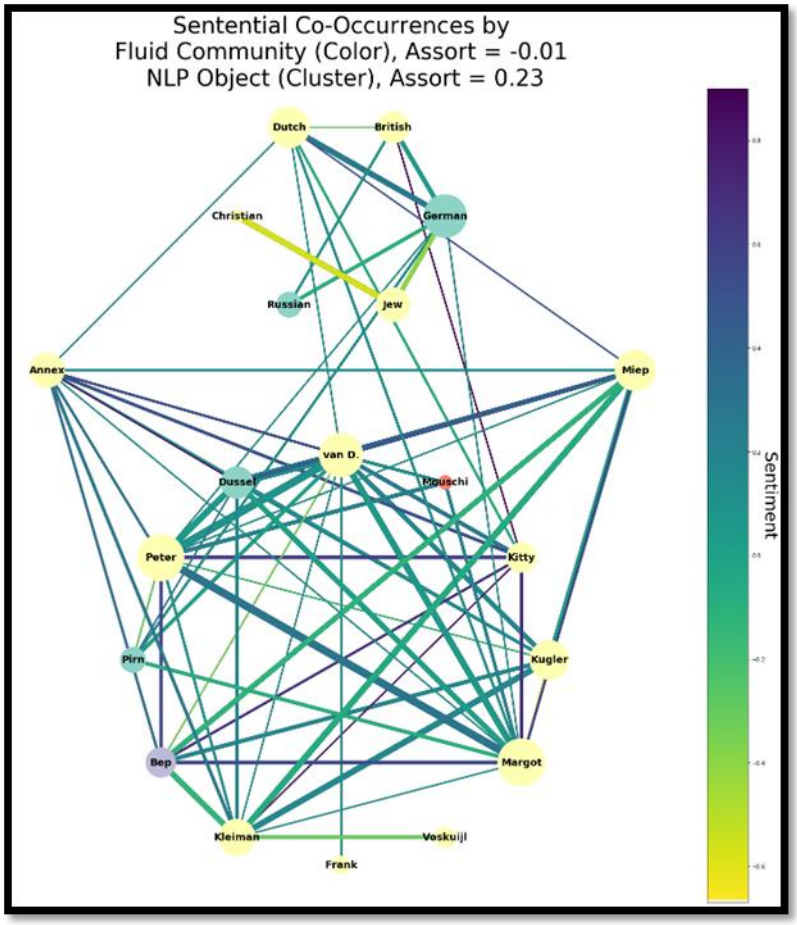

Figure I4: Sentential Co-Occurrence for Frequent Terms by NLP Attribute (Fluid Coloring)

Figure I5 shows sentential co-occurrences clustered by NLP Attribute with the nodes now colored by the Louvain community. The edges between Dussel & van D. and Peter & van D. now appear to be the contributors of the significant edge between Louvain Community 0 &

Louvain Community 2. Peter was, of course, of the van Pels family (aka van Daan) whom Anne was both fond of and mentioned frequently. Previously, this project explored the slope in sentiment between Dussel & the van D. family in 1943. Though their relationship deteriorated over time, the sentential co-occurring sentiment for all years can be seen to be a net positive. This average sentiment comes from a mix of strongly positive sentiment that caused relations to worsen that same year, most notably an affair, “Mrs. van D. sprang out of bed and went downstairs to Dussel's room to seek the comfort she was unable to find with her spouse.”

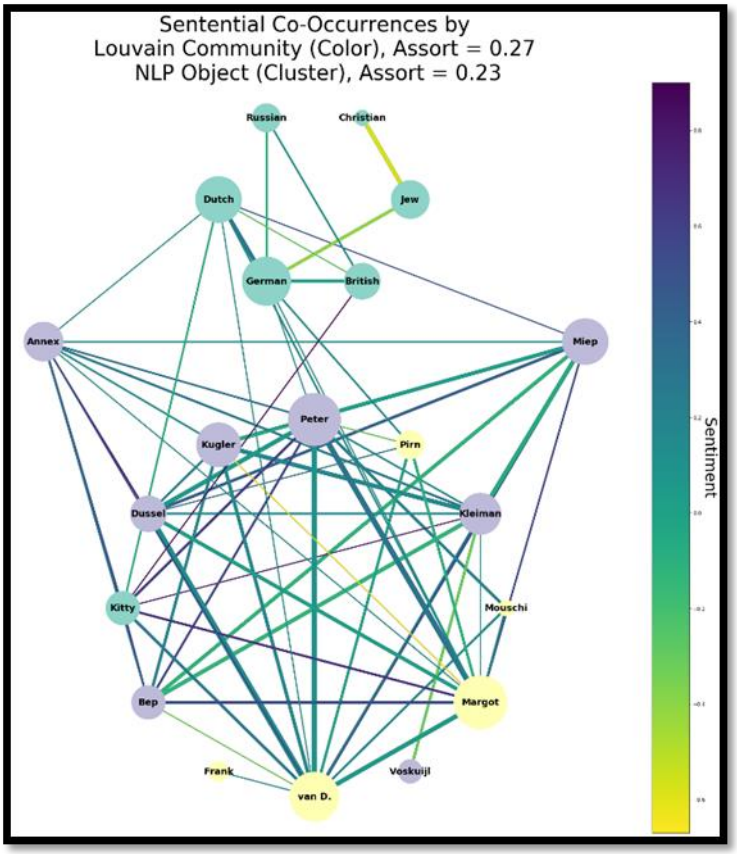

Figure I5: Sentential Co-Occurrence for Frequent Terms by NLP Attribute (Louvain Coloring)

All the NORP’s are now grouped together except for Kitty. The remaining Louvain communities differ over Product/Person/ORG. The Louvain method is unlike the NLP community detection method in that it does not consider the context of the words in the

detection of proper nouns. The Louvain method simply takes the node & edge attributes of the already detected proper noun and iteratively finds the optimal community based on the pre-defined algorithm. It is the overlap of contextual community detection (NLP) with context-free modularity in the top cluster of Figure I5's NORP's that shows network theory accuracy from these two separate community approaches.

Performing the same graph except clustering by Louvain Community and coloring by NLP attribute is shown in Figure I6. Van D. can be seen more clearly responsible for the higher average edge width during summarization between the two Louvain communities comprised predominantly of Annex residents. In addition to Kitty not being a NORP, the degree Kitty has with differing Louvain communities also appears to stand out. Unlike the Louvain summarization showing positive sentiment between communities, a more nuanced picture emerges after showing the relationship between proper nouns within each Louvain community. The community containing mostly NORP's has the highest amount of negative sentiment as expressed between the terms Christian & Jew and German & Jew. The Louvain community containing Annex has overall positive sentiment driven by Miep and the Annex itself. Family names and alternate names for family members make up the final Louvain community containing van D. The latter community is expectedly more interconnected to the Annex community than the community densely populated with NORP's.

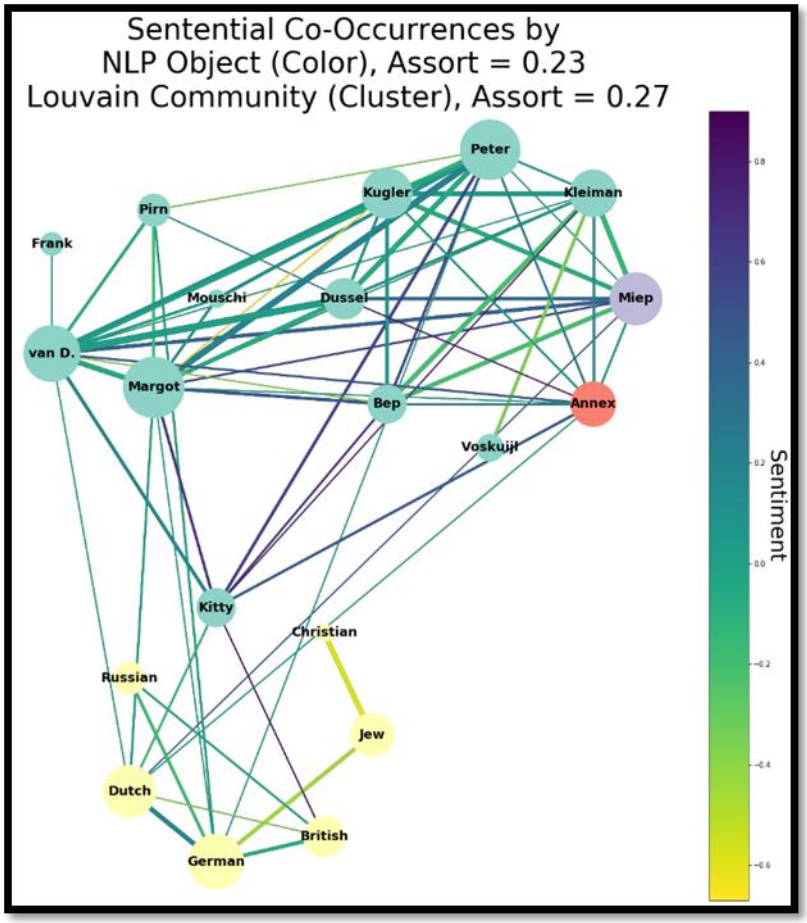

Figure I6: Sentential Co-Occurrence for Frequent Terms by Louvain Community

In Figure I7, the same graph but only for 1944 shows a higher assortativity using NLP/Louvain method community detection. This year was used due to the higher mentions of NORP's relative to mentions of members of the Annex. The Louvain algorithm finds four distinct communities for this year best described as NORP's, Annex members, Annex helpers, & Anne's friends (Peter, the cats, and the radio). The Louvain assortativity for 1944 is both better than NLP assortativity in 1944 as well as both technique's assortativity for the overall diary. In other words, the intra-community edges are best explained by the Louvain method for 1944 more so than the algorithm can explain sentential co-occurrence for the diary.

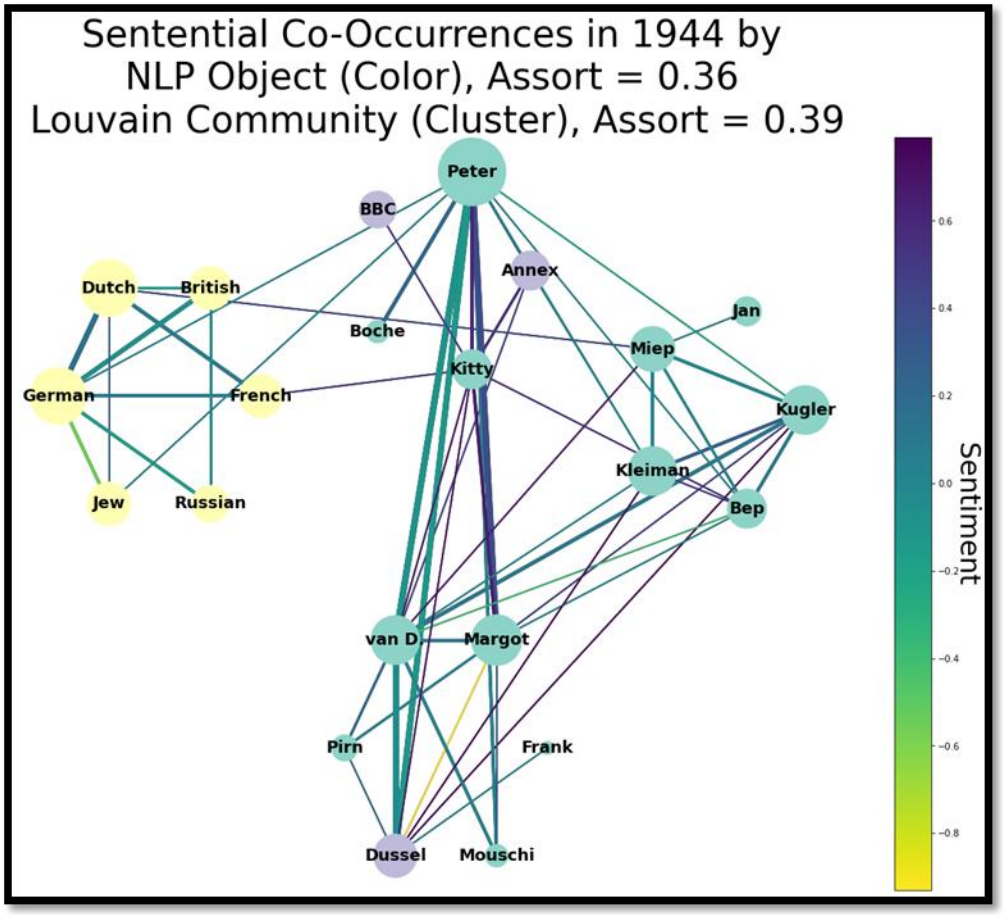

Figure I7: Sentential Co-Occurrence in 1944 for Frequent Terms

Judging by their assortativity, it is possible that as Anne grew more discontent with the progress of the war, her reflections on the frequent terms (NORP's, her friends) helped solidify natural relationships more easily detectable by NLP/Louvain communities. On May 16, 1944, there are two examples of how this might be happening, "Peter van Daan. Is learning English, French (correspondence course), shorthand in Dutch, English, and German, commercial correspondence in English, woodworking, economics and sometimes math; seldom reads, sometimes geography." Then, shortly after that entry, "Anne Frank. Shorthand in French, English, German and Dutch, geometry, algebra, history, geography, art history, mythology, biology, Bible history, Dutch literature; likes to read biographies, dull or exciting, and history

books (sometimes novels and light reading).” By describing the languages each member of the Annex is learning in separate sentences, this entry is contributing to intra-community edges within the NORP Louvain community while building fewer edges exterior to that community back to the members of the Annex.

Though sentiment between terms in 1944 was presented earlier in this project, it was without being able to discuss the general sentiment between detected communities. Figure I8 presents the summarization of the sentiment between Louvain communities in 1944. Figure I9 shows the membership of each Louvain community in 1944. All communities appear to relate to positive sentiment except for that of Louvain Community 0, the NORP’s, and Louvain Community 1, the group of Annex members that Anne did not consider her close friends. In the year 1944, the diary has no sentential co-occurrence between any NORP and Frank, Margot, van D., Pirn, Dussel, or Mouschi. Interpretations of this fact may vary, but one thing is apparent. Towards the end of 1943, specific entries indicate apprehension at discussing doubts about the progress of the war publicly:

“Mrs. van D. casts about for another topic. ‘Tell me, Putti, why aren't the British carrying out any bombing raids today?’”

"Because the weather's bad, Kerb!"

"But yesterday it was such nice weather and they weren't flying then either."

"Let's drop the subject."

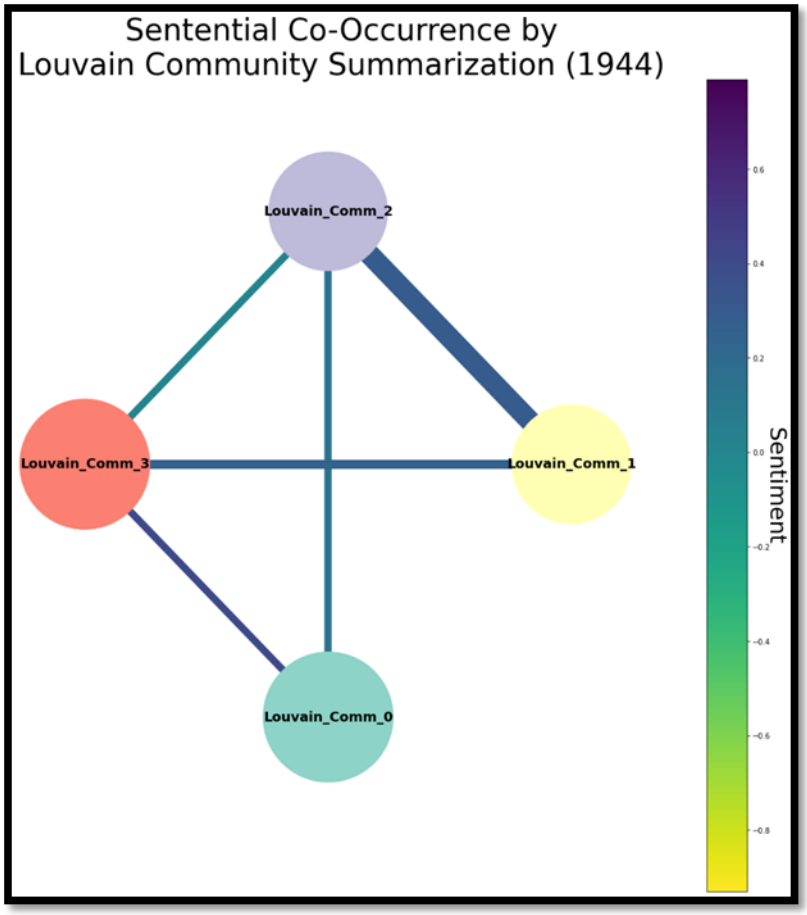

Figure I8: Sentiment of Sentential Co-Occurrence by Louvain Attribute (1944 Summarization)

```
Louvain Community 0 contains ['French', 'British', 'Dutch', 'German', 'Jew', 'Russian']
Louvain Community 1 contains ['Frank', 'Margot', 'van D.', 'Pirn', 'Dussel', 'Mouschi']
Louvain Community 2 contains ['Annex', 'Peter', 'BBC', 'Boche', 'Kitty']
Louvain Community 3 contains ['Kugler', 'Jan', 'Miep', 'Kleiman', 'Bep']
```

Figure I9: Louvain Community Membership for 1944

Edge width from this Louvain community summarization from 1944 shows co-occurrence was highest between the two communities containing members of the Annex. The sentiment of this edge is mainly positive. More positive still is the edge between Louvain Community 0, the NORP terms, & Louvain Community 3, the Annex helpers. Without summarization, some of the most positive sentiment exists between communities rather than

within Louvain communities, while some of the most negative sentiment exists between NORP's. To further explore the sentiment surrounding nodes in 1944, Figure I10 shows the weighted sentiment per node for Anne's final year in the Annex.

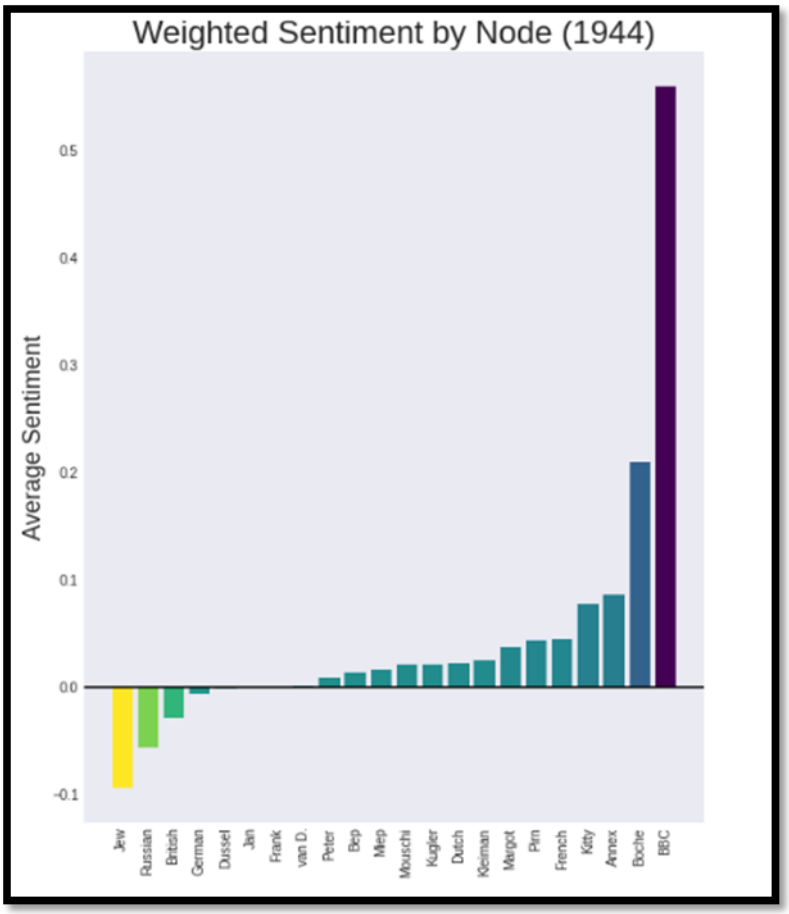

Figure I10: Weighted Sentiment by Node in 1944

One might expect the majority of terms to be surrounded by neutral to negative sentiment from an author who had spent so much time in hiding. The findings of this experiment were the opposite. Most nodes are surrounded by positive sentiment, excluding Dutch & French. BBC, Boche, Annex, & Kitty, which made up most of the Louvain community most closely associated with Anne's friends. These were all found to be terms surrounded by

the most positive sentiment. A mix of both Annex residents and helpers of the Annex occupies the remaining positive sentiment.

h.Comparison of Three Community Algorithms

In order to adequately baseline results from Fluid and Louvain community detection methods, the Leiden community detection algorithm was employed. The Leiden algorithm differs from Louvain by randomly moving local nodes to nearby communities to find the best partition (Traag, Waltman, & Van Eck, 2019). Figures J1, J2, & J3 show the three different community detection methods on the sentential co-occurring frequent terms.

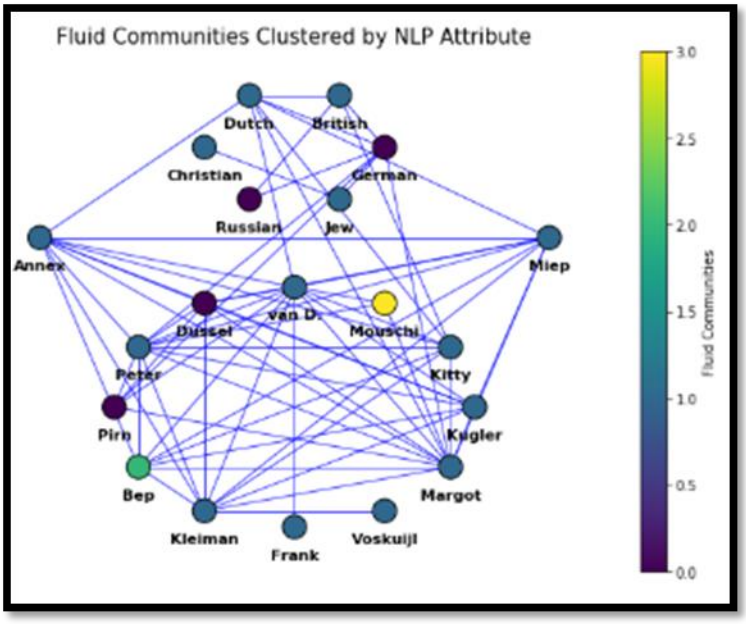

Figure J1: Fluid Communities Clustered by NLP Attribute

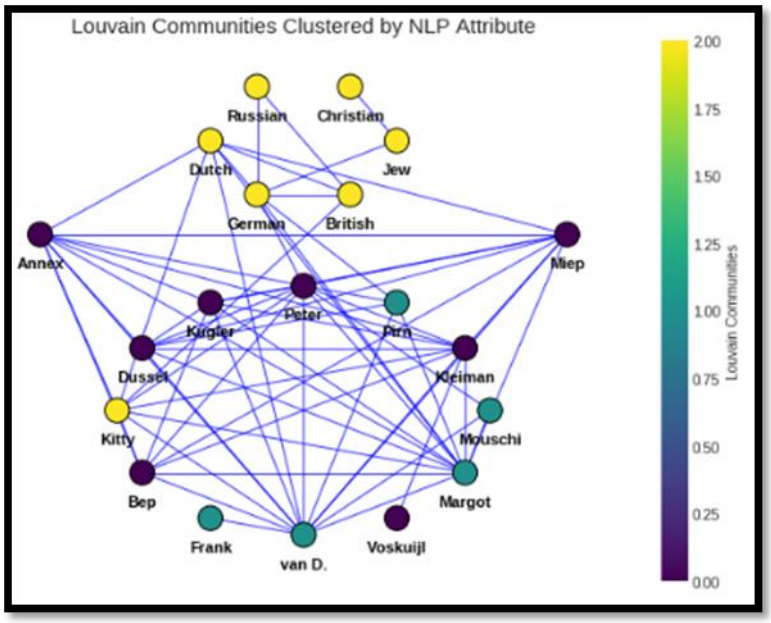

Figure J2: Louvain Communities Clustered by NLP Attribute

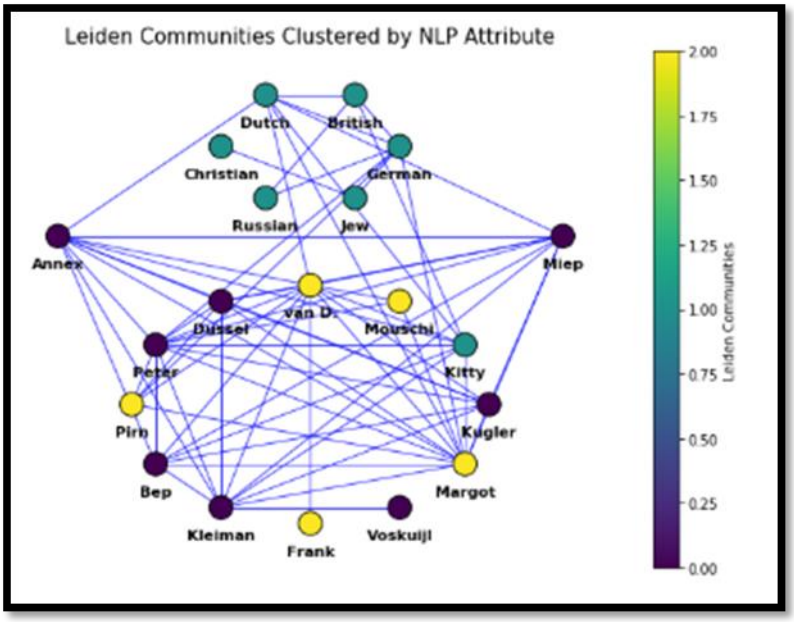

Figure J3: Leiden Communities Clustered by NLP Attribute

Figure J4 below shows all communities via each detection method as nodes and the similarity of terms between each community as weighted edges. Three highly weighted edges exist between individual Leiden and Louvain communities. Since each Louvain community has a

Leiden community it directly correlates to, both of these detection methods obtained the same membership. Fluid Communities have very low weights across all edges and share very little membership in common with the Leiden and Louvain community detection methods.

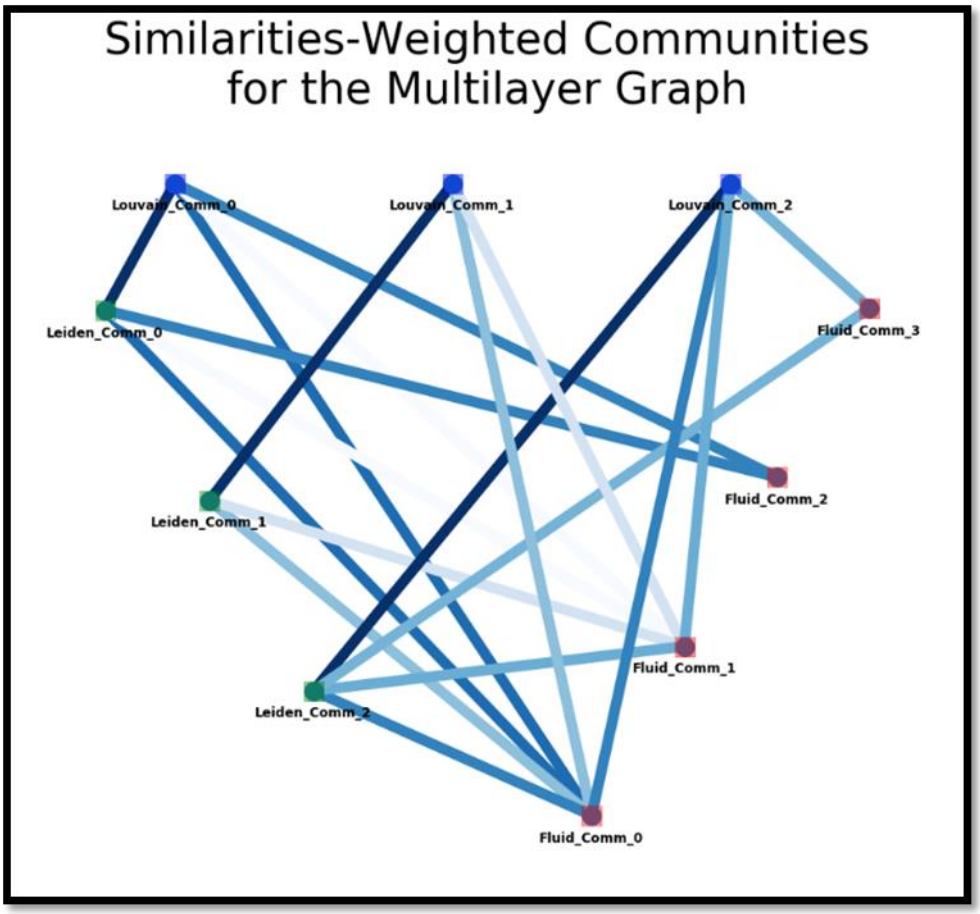

Figure J4: Similarity-Weighted Communities for Fluid, Louvain, & Leiden Methods

The same behavior can be seen even with less data when looking at individual years. Leiden communities & Louvain communities always have strongly weighted edges between each other, though focusing on individual years often found more communities through these methods. Figure J5 shows the same figure again for 1944 with strongly weighted edges between Louvain & Leiden communities. The effect of increasing Louvain & Leiden communities resulted in more uniform edges to the fluid communities. That result gives some

evidence that fluid community detection may have been more random in its approach to community detection. The Leiden & Louvain methods may share community detection precision while still performing overall worse than fluid community detection. For that reason, the overall performance of community detection techniques was explored for this project.

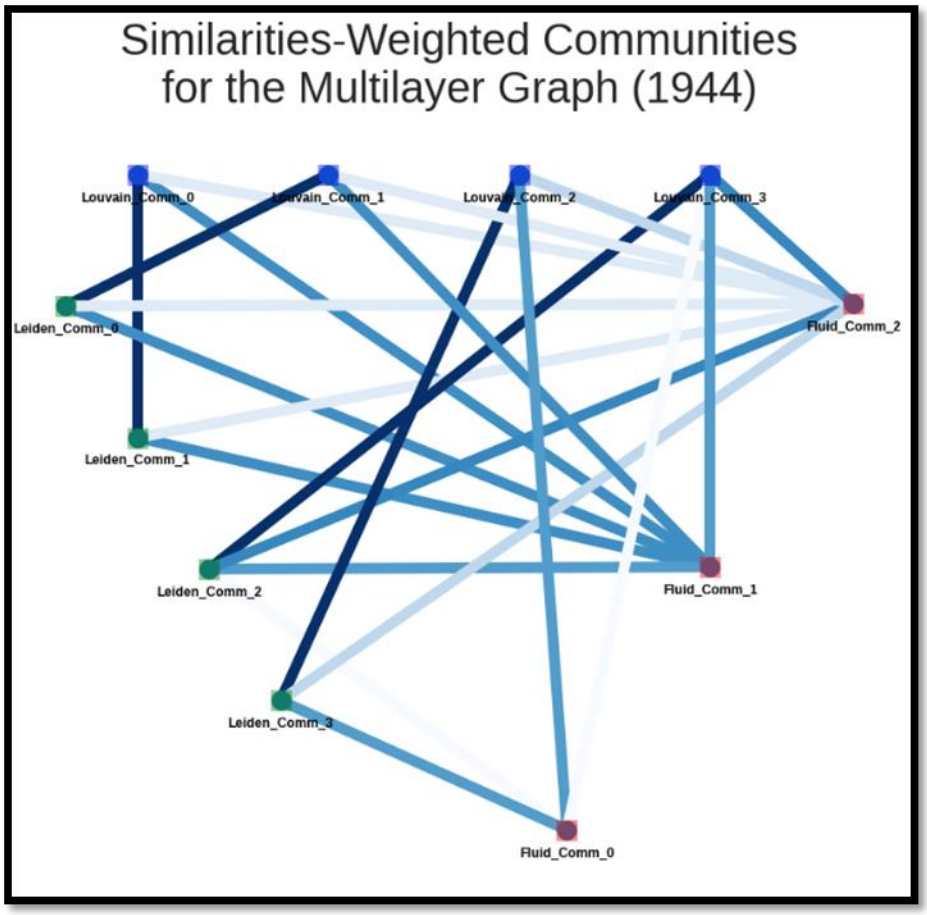

Figure J5: Similarity-Weighted Communities for Fluid, Louvain, & Leiden Methods (1944)

Figure J6 below shows the metrics of these community detection methods. Performance is the number of intra-community edges plus intra-community non-edges with the total number of potential edges. Modularity compares the number of edges inside a community to that of a network with randomly distributed edges (a poor assumption for vast networks). Coverage is the ratio of intra-community edges to the total number of edges in the graph. Fluid

detection having negative modularity means the fluid communities had fewer edges between vertices of the same community than one would expect by chance. Since Leiden & Louvain communities have the same membership, these two detection algorithms were matched in all metrics. Fluid community detection had the highest coverage of all detection methods meaning it retained the most intra-community edges versus overall edges in the graph.

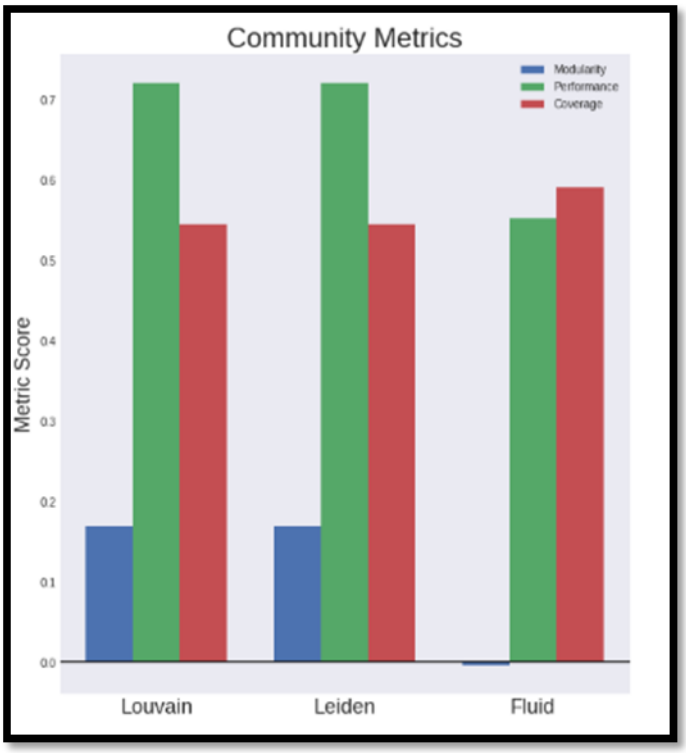

Figure J6: Community Metrics of the Fluid, Louvain, & Leiden Methods

Specific community detection techniques (Multilevel/Eigenvector/FastGreedy) rely primarily on modularity maximization. The issue with these methods is that they can result in unrealistically large communities since a broader community mirrors that of increased samples from a distribution. More nodes captured in a community in this maximization technique are likely to increase the similarity of a community with random edges (Lancichinetti & Fortunato, 2014). On the other hand, it is still vital for modularity to be a factor in identifying the proper

community detection algorithm since each community should mirror randomly distributed edges. Performance evaluates edges connecting intra-community vertices as well as the edges that do/do not exist between vertices of differing communities (the lack thereof being desirable from the algorithm). Coverage is then, put another way, the ratio of edges that remain intra-community over those that are between communities. Since the Fluid community algorithm maximized the ratio of intra-community edges but failed to detect communities that matched those of randomly distributed edges, the Louvain detection method was chosen since it is one of the standard community detection algorithms in graph theory.

i.Cliques

To further look at how the nodes naturally cluster together, cliques were created to explore like-term membership. Figure K1 shows the clique membership of nodes. Nodes that seemed more central to many of the graphs, namely Margot/Peter/van D./Kleiman, all have some of the most considerable degree for clique membership. Most cliques contain at least three or more nodes.

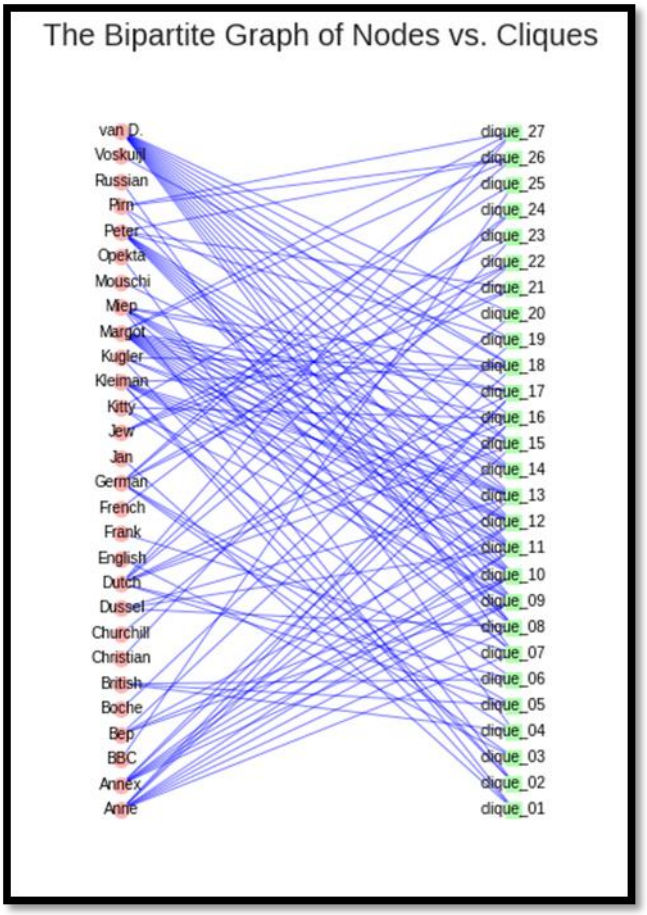

Figure K1: Term Membership in Cliques

Figure K2 imagines what a similar graph would look like except as the intersection of these cliques as nodes and the membership of shared terms as edges. The shared terms between cliques 7 through 17 are even now more apparent as a cluster in the bottom right of this graph. Comparing the membership of which outlier cliques contain which terms, most of the NORP's and those who assisted the Annex reside in these satellite cliques. Satellite clique membership makes sense since they played less of a direct role in Anne's day-to-day life and primarily affected the conditions inside and surrounding the Annex. Certain cliques outside the cluster are still closer than most, most notably Clique 1, perhaps in part due to the membership

of the term German in that clique and the more direct relevance it had to the residents of the Annex.

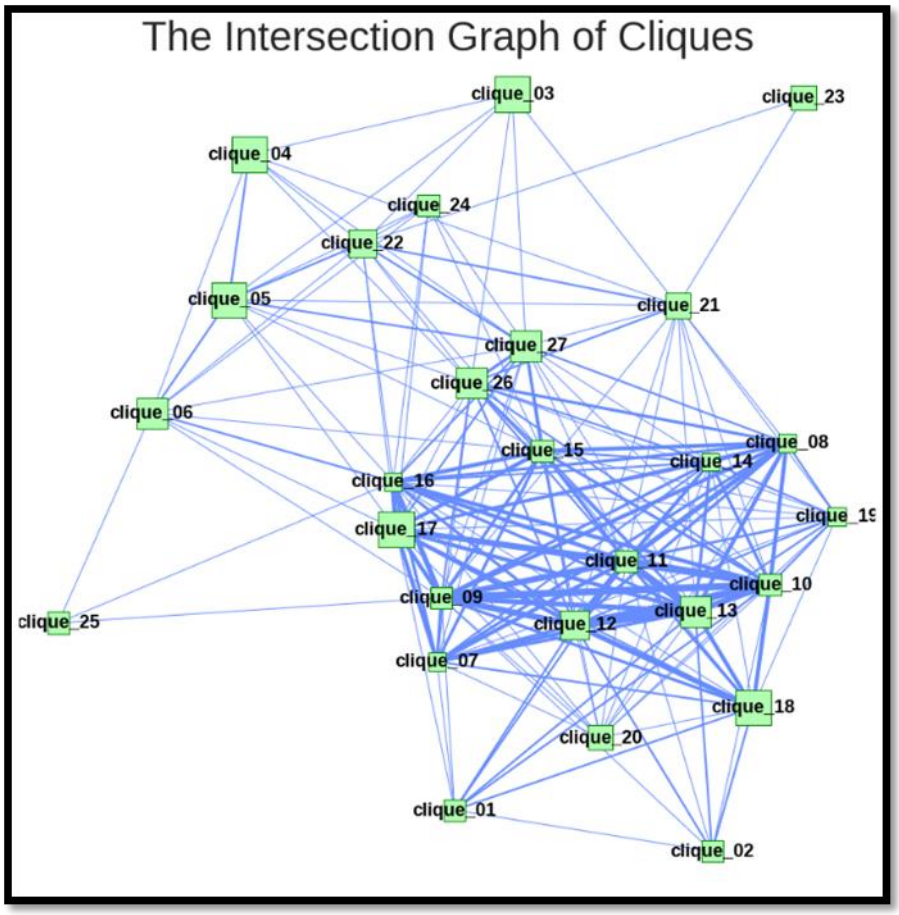

Figure K2: Intersection of Cliques

To apply this knowledge of how cliques intersect to the original graphs of terms as nodes, Figure K3 shows a graph of how each term relates to each other with the clique intersection type as edges. Here, orange nodes represent leaves, red nodes represent bridges between cliques, cyan edges represent non-intersecting clique edges, and blue edges represent intersecting clique edges. French, English, and other NORP's that had less direct influence on the residents of the Annex remain as the leaves with less connectedness within the cliques. German, Dutch, & other NORP's + helpers that played more of a direct influence act as bridges

between the cliques while retaining a relatively low degree to as many cliques. Finally, Margot, Dussel, & other residents of the Annex serve as both bridges between both intersecting & non-intersecting cliques as well as display a high degree of edges with many of the cliques that exist.

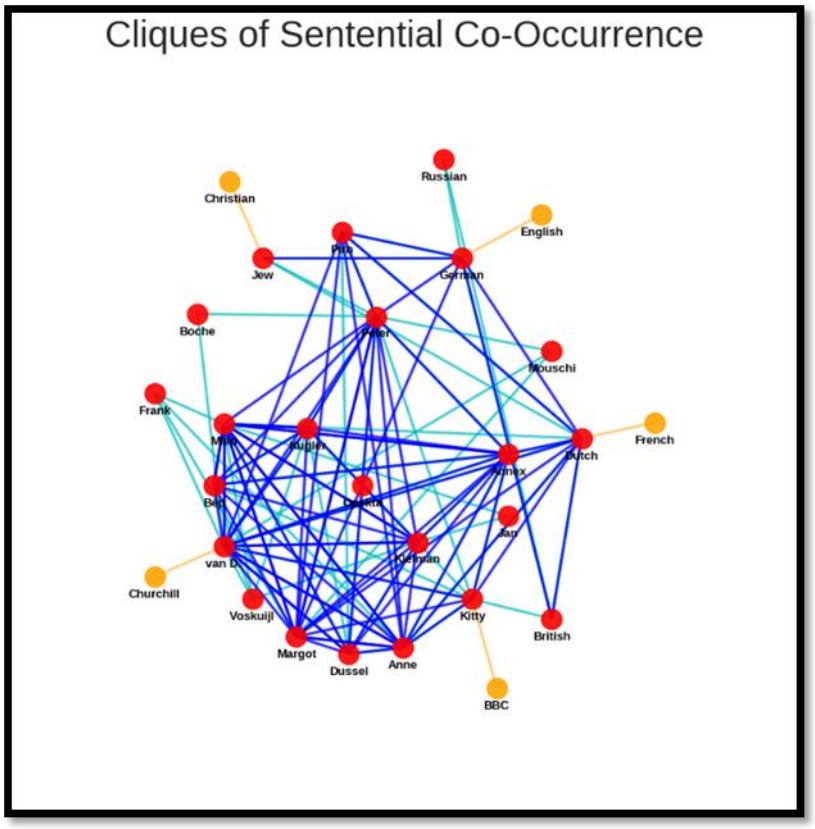

Figure K3: Clique Bridging by Term

Looking at clique evolution temporally, a different type of node emerges in Figure K4 when looking at clique bridging for 1944. The green nodes below represent brokerage between cliques. Terms with this behavior in cliques bring sharply divergent factions closer together (Gabbay, 2007). French & Jew fulfill this role and makes sense in the context of previous findings of this research. French appears in the diary both in use as a language & to discuss external relations between NORP's. The term Jew appears not only in relation to NORP's but also in relation to members of the Annex discussing the perseverance of the Jewish people,

“Peter added, ‘The Jews have been and always will be the chosen people!’” Both act as bridges to otherwise disconnected cliques within the context of 1944’s entries. The overall clique bridging for all years showed both Jew & French in red since the cliques they connected were not otherwise divergent and had other edges between the terms connecting these cliques.

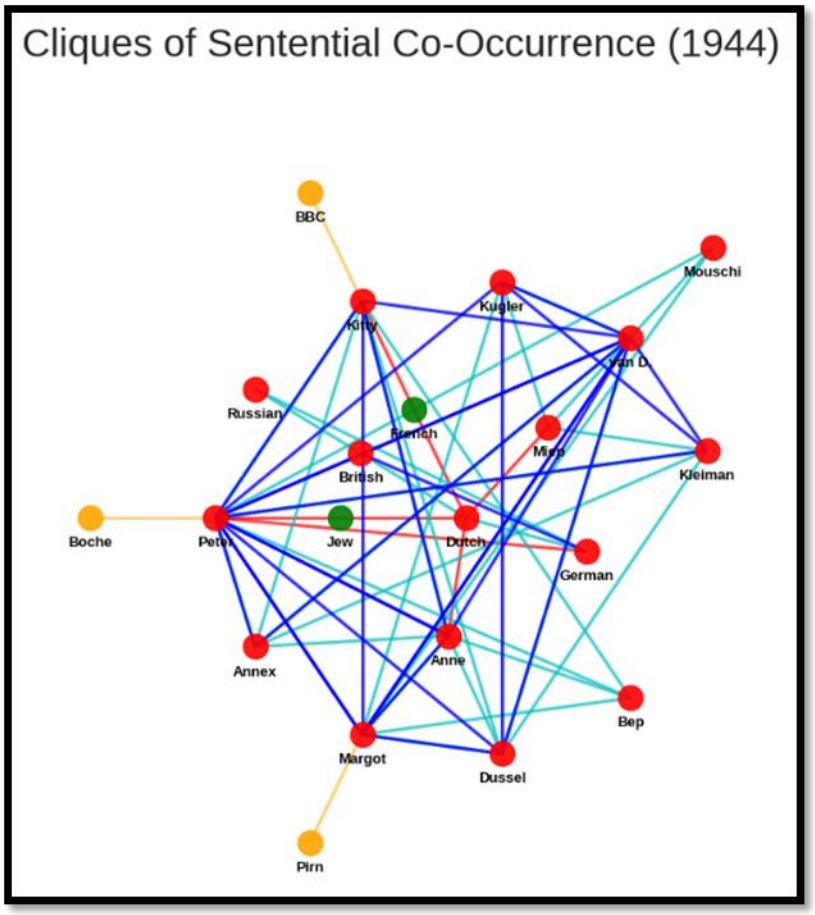

Figure K4: Clique Bridging by Term (1944)

j.Centrality Indices & Sentiment Distribution

To better compare the centrality measures of each term, Figure L1 shows the rankings of each node for the overall centrality metrics (Freeman, 2004; Newman, 2019). Margot, Peter, & van D. all appear at the highest rank overall. Although previous analyses of the graphs

discussed nodes/cliques that had a high measure of centrality regarding closeness to other nodes, this rank factors in nodes that served essential relationships such as betweenness and load. In this way, a node that was not one of the closest or part of Anne's circle in the Annex could still be ranked highly due to its overall relationship to the proper nouns of the diary, e.g., Dutch rank is 2.

node	rank	degree	closeness	betweenness	eigenvector	HITS	Katz	PageRank	load	communicability	current flow
van D.	1	8	7	4	16	16	14	9	4	14	14
Margot	1	6	4	5	11	11	11	8	5	10	9
Peter	1	14	17	17	28	28	28	24	17	28	28
Dutch	2	5	2	3	12	12	10	6	3	8	7
Anne	3	13	13	17	22	22	22	22	17	22	22
Kleiman	3	10	10	17	14	14	15	17	17	15	15
Miep	3	4	3	11	5	5	6	7	12	6	6
German	4	3	3	7	6	6	5	4	7	5	5
Kitty	4	1	1	1	1	1	1	1	1	1	1
Annex	4	11	10	12	21	21	19	16	11	19	17
Bep	5	6	5	15	8	8	8	12	15	9	10
Kugler	6	12	9	17	17	17	17	19	17	18	19
Pirn	7	3	3	9	4	4	4	5	9	4	4
Dussel	7	13	12	17	20	20	21	23	17	21	21
Jew	8	2	1	2	3	3	3	2	2	3	2
Frank	9	9	7	13	13	13	13	14	13	13	13
Mouschi	9	5	4	14	7	7	7	10	14	7	8
Voskuijl	10	14	11	17	23	23	24	28	17	23	24
British	10	14	13	17	25	25	25	26	17	25	25
Boche	10	13	10	17	18	18	20	21	17	20	20
Churchill	11	13	15	17	26	26	23	20	17	24	23
Jan	12	11	9	17	15	15	16	18	17	17	16
Opekta	13	6	6	10	10	10	9	11	10	11	11
French	13	7	7	16	9	9	12	13	16	12	12
Russian	13	14	14	17	24	24	26	27	17	26	26
Christian	14	11	8	8	19	19	18	15	8	16	18
BBC	14	14	16	17	27	27	27	25	17	27	27
English	14	2	1	6	2	2	2	3	6	2	3

Figure L1: Node Rank by Measures of Centrality

Figure L2 reimages this aggregate centrality ranking, with nodes like Dutch & Annex showing more considerable differences. Dutch not only is central to the members of the Annex but serves as a bridging node to specific NORP's. This bridging is due to the duality of this specific term. It both describes events surrounding the Annex as well as objects brought into the Annex.

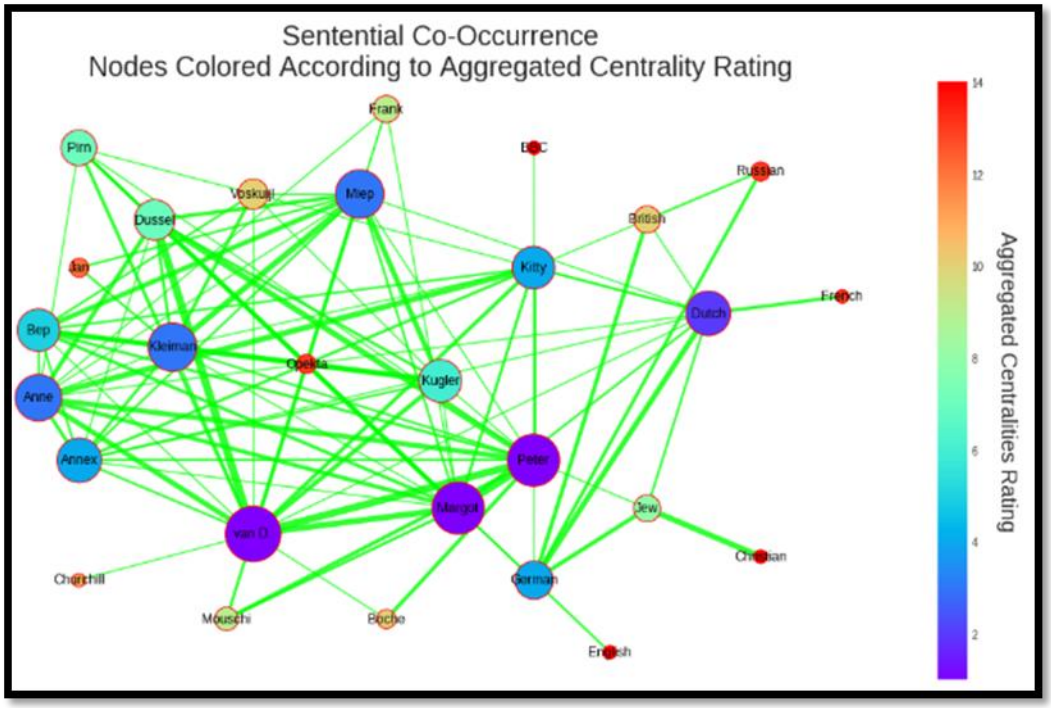

Figure L2: Node Rank by Measures of Centrality

Figure L3 below shows the scatter matrix of the centrality rankings per node. The horizontal line across the matrix where nodes mix with themselves shows the distribution of rankings. Note how Kitty's cell for this distribution is highly skewed to the left due to that term's high measures of centrality. In the diary, Anne imagined Kitty as her best friend instead of having friends outside the Annex, making Kitty central to the text in different metrics. Looking at the scatter matrix rows for Anne & Kitty, Christian & Jew, and German & Dutch show some degree of linearity, implying there is a relationship in measure of centrality. Previous graphs involving strong sentiment and high co-occurrence have given other aspects to the story behind the centrality metrics of these pairs.

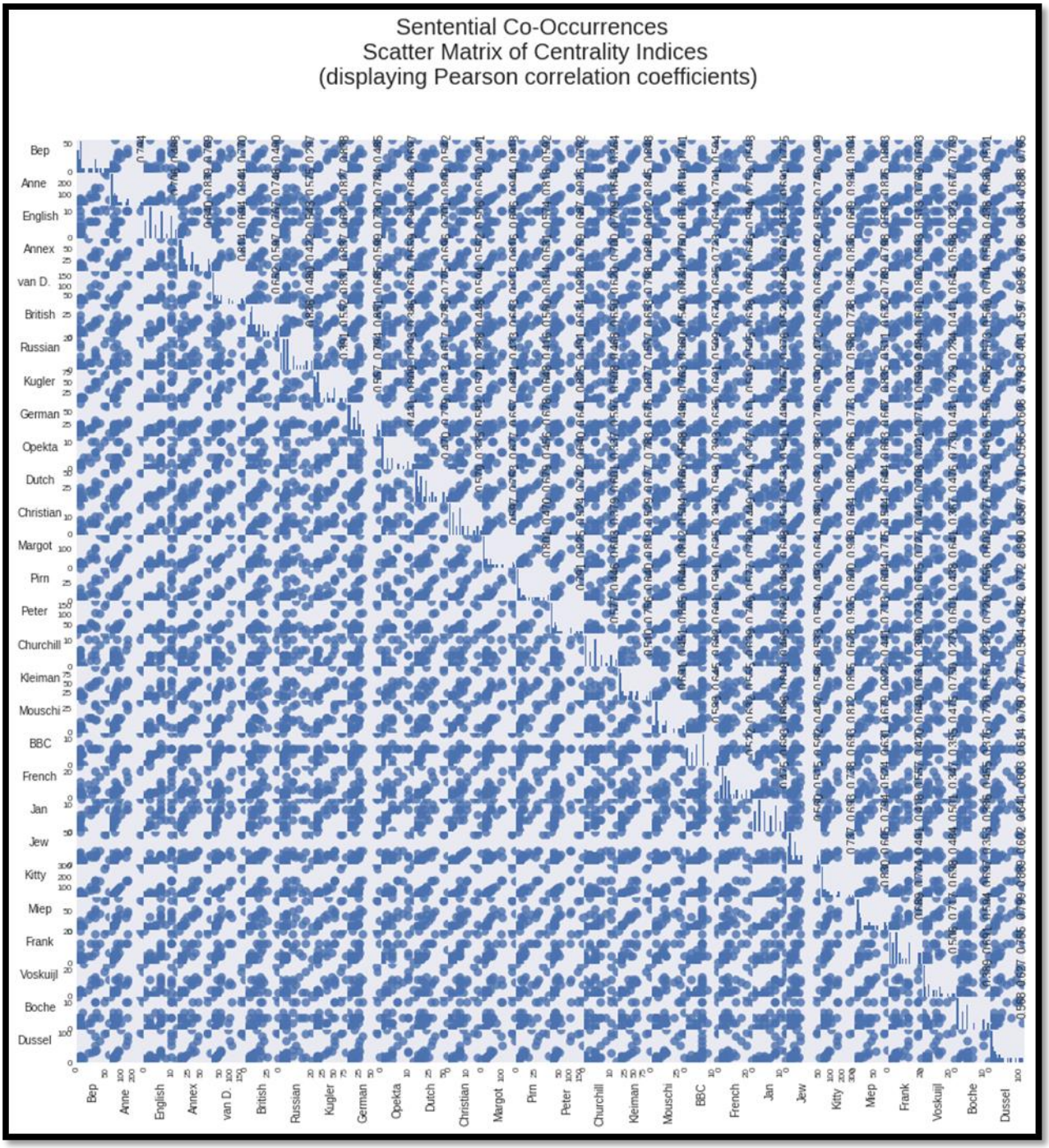

Figure L3: Scatter Matrix of Centrality Indices with Pearson Correlation Coefficients

Comparing different methods of measuring centrality, Figure L4 shows the Katz centrality of each term. Here, node size and darkness represent terms with higher measures of Katz centrality. Katz centrality measures the number of neighbors to a node as well as distant

neighbors. The more distant a neighbor, the less that neighbor counts towards a given node's Katz centrality. Katz centrality's usefulness stems from its ability to gauge the relative nearness of neighbors in a given network. Van D., Anne, & Peter all perform highly for this metric as one might expect. One node to notice that performed highly for this metric, interestingly, is Kleiman. As Anne notes, "When Mr. Kleiman enters the room, the sun begins to shine!" Kleiman might, in part, have a high Katz centrality measure due to becoming known to make cheering everyone up as his goal. With these actions and Anne noting them, the Kleiman node interacts with more neighbors than other assistants or NORP nodes.

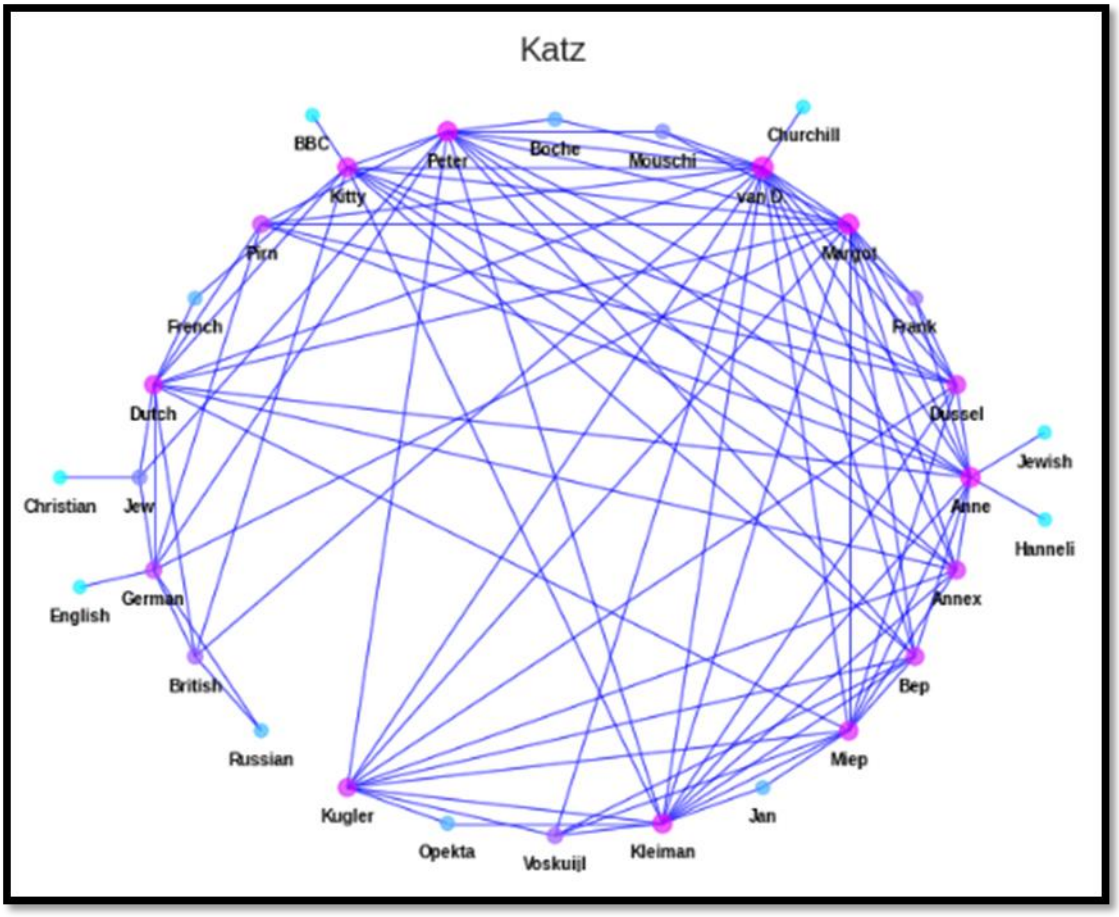

Figure L4: Katz Centrality by Node

Figure L5 shows a depiction much like the former except using communicability as a centrality measure. Communicability is the sum of closed walks of different lengths starting at node U and ending at node V. Anne ranks even higher in this overall centrality measure than Katz. Benzi & Klymko (2013) describe communicability as a measure of how easily information flows between two nodes. The definition by Benzi & Klymko implies that although Anne has relatively less centrality concerning her immediate & distant neighbors, she was more critical to the network in the exchange of information. Jewish, Christian, English, and other nodes with relatively low Katz centrality rankings are not just relatively lowly ranked by communicability but also have an almost nonexistent communicability metric. Whereas Katz centrality considers distant neighbors for nodes with few connections, communicability will see almost nonexistent possible walks due to those few connections.

It is of interest that Jewish & Christian both have one of the lowest sentential co-occurrent sentiment while also being ranked lowest for the centrality ranking that controls the flow of information between nodes. These two rankings may be a result of the author using writing to focus less on negativity. As when she writes, “My writing has raised me somewhat from ‘the depths of despair.’” Nodes involved in negative sentiment incur lower measures of certain types of centrality. In the case of communicability, this is true for English & German, Jewish & Anne, and Hanneli & Anne. Hanneli was an old classmate of Anne’s that she missed dearly and thus often associated with negative sentiment.

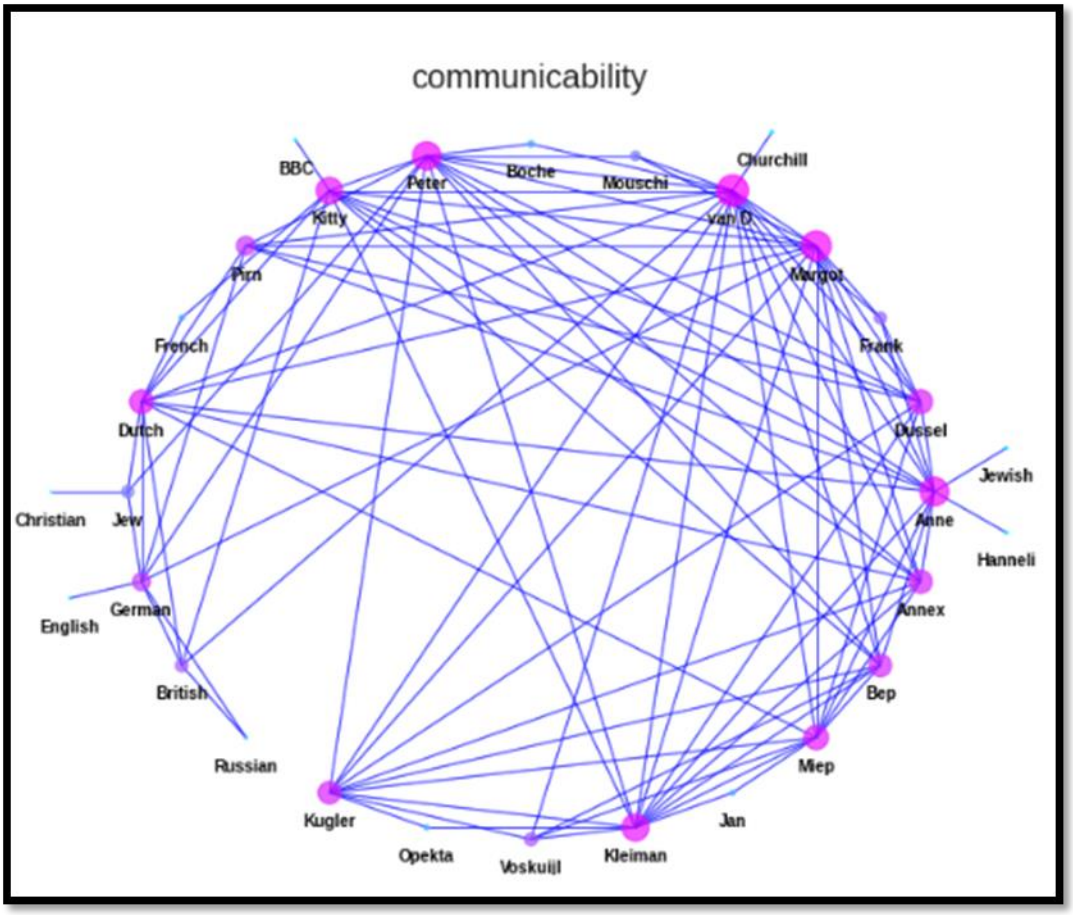

Figure L5: Communicability Centrality by Node

To test this theory, Figure L6 shows the relationship between communicability and the average weighted sentiment around each term. Communicability does not have a strong linear correlation with sentiment. More so, a statistically insignificant p-value of 0.38 resulted from the Pearson correlation coefficient. Instead, terms with higher communicability center around neutral sentiment. Since the sentiment of the y-axis represents the collective sentiment around each node, as terms are involved in more of the exchange of information, it is possible that negative & positive sentiment in this exchange result in a neutral effect. Terms less involved in the exchange of information are mostly NORP's with an outlying sentiment.

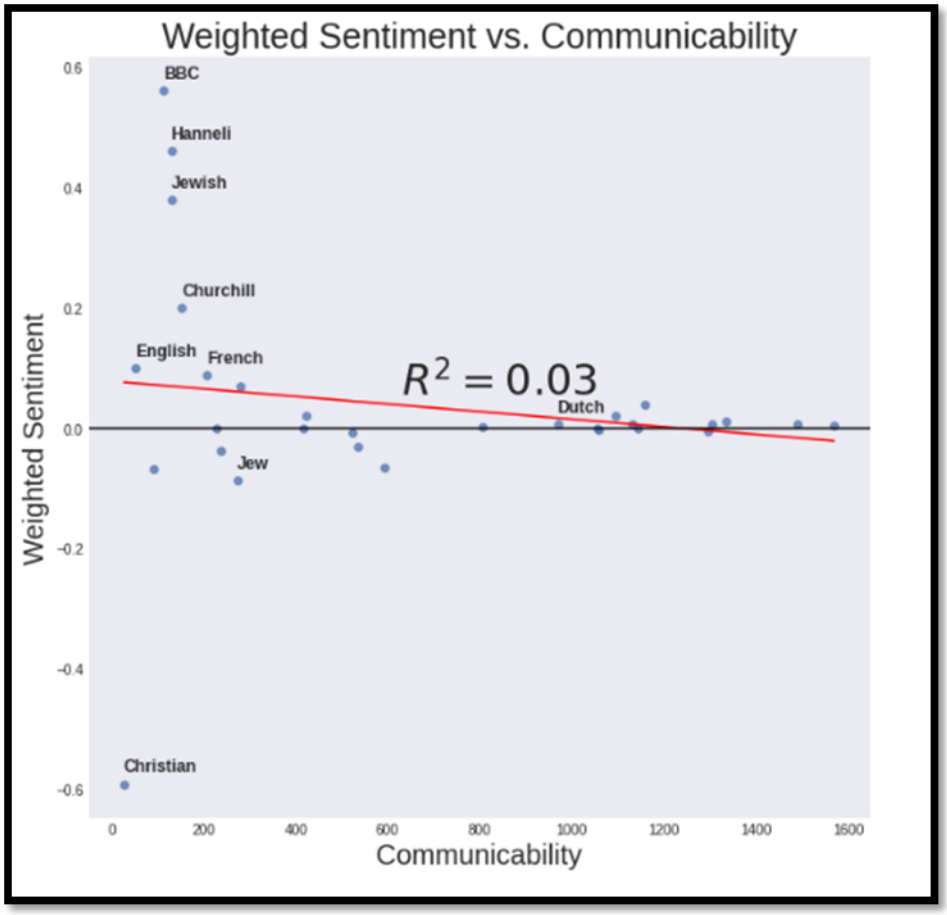

Figure L6: Relationship Between Weighted Sentiment & Communicability

This behavior was noticed for other measures of centrality, including closeness, as shown by Figure L7. With a lower R^2 value, the relationship between closeness and sentiment acts even more non-linearly than communicability. Like communicability, closeness is the measure of the inverse distance of all nodes relative to a given node. Closeness centrality also appeared to have the same net effect on the average sentiment of nodes with a higher measure of this centrality. The same NORP terms appear to be outliers with strong sentiment and low centrality rankings.

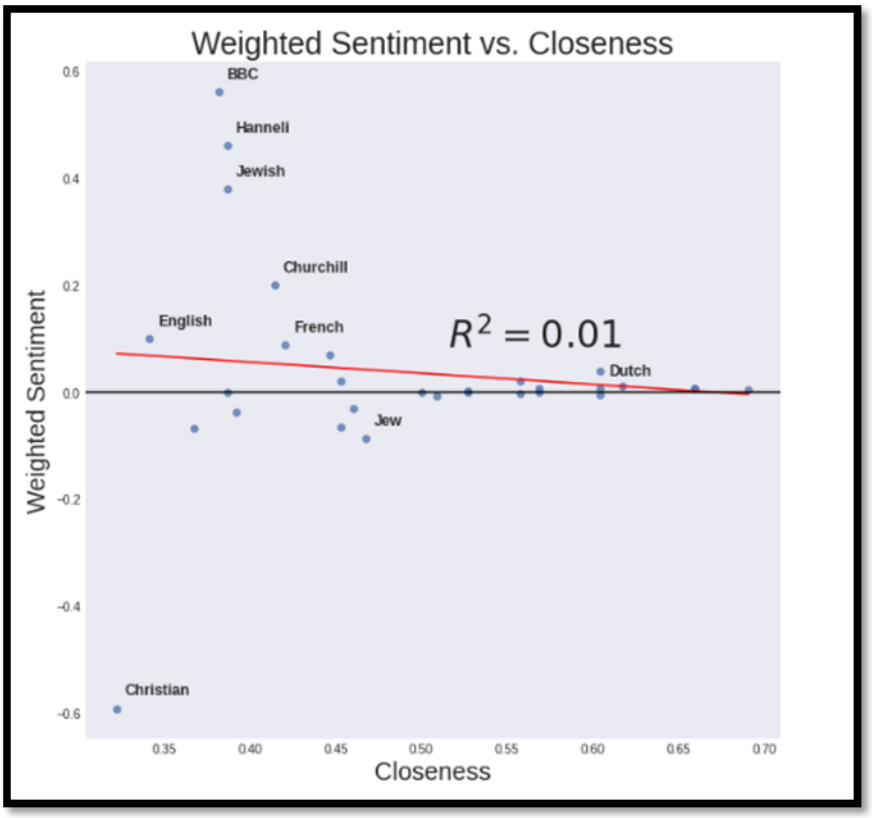

Figure L7: Relationship Between Weighted Sentiment & Closeness

In contrast with communicability, load centrality is the fraction of all shortest paths that pass through a given node. Figure L8 shows the relative load centrality metrics by node. Since tertiary nodes with one edge are involved in no shortest paths, their load centrality is zero. A load centrality score of zero is a steeper penalty for many of the NORP's relative lack of edges than other measures of centrality. Unlike Katz centrality where van D. had comparable centrality to other nodes, van D. is by far one of the few nodes with a high load centrality metric. Nodes like Annex, with high Katz centrality and low load centrality, may have a lot of neighboring nodes but are rarely the shortest path of those neighbors, unlike van D.

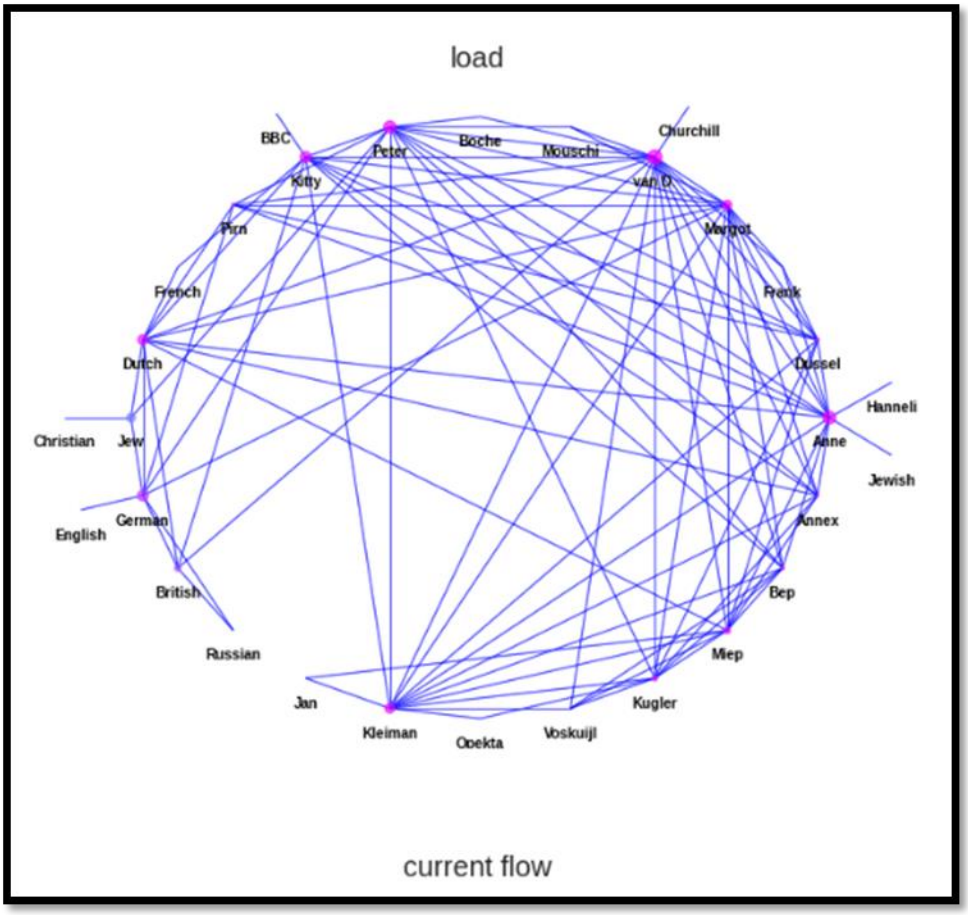

Figure L8: Load Centrality by Node

Earlier in this project, part of this lack of edges in NORP's was found to be due to a total absence of sentential co-occurrence in 1944 between discovered Louvain communities. Terms like van D., Peter, Anne, & Kitty that remained consistently interconnected throughout most entries every year seemingly retain the ability to establish being highest in the fraction of shortest paths. Also, since load centrality uses shortest paths, it had the effect in Figure K8 of giving only one or two terms per Louvain community the high load centralities while giving the rest of the nodes in those communities near-zero scores. Looking at the term Kleiman, for example, the node has a relatively high load centrality due to its edges with Peter, Kitty, & van D., or other nodes with high load centrality. Looking at other helpers of the Annex, they lack

these edges with the other high-load nodes and thus have almost nonexistent measures of load centrality. Their exclusion in load is due to measures of the shortest path more often involving the term Kleiman.

Figure L9 shows the effect of the shortest path criteria in load centrality. Like previous centrality measures, nodes with lower measures of centrality still have higher variance in sentiment and vice versa. More extensive clustering of nodes around a zero centrality measure with a quicker centering of nodes around zero sentiment beyond that region shows the steepness in penalty is more significant for the load centrality. There remains a high degree of non-linearity between weighted sentiment and centrality measures.

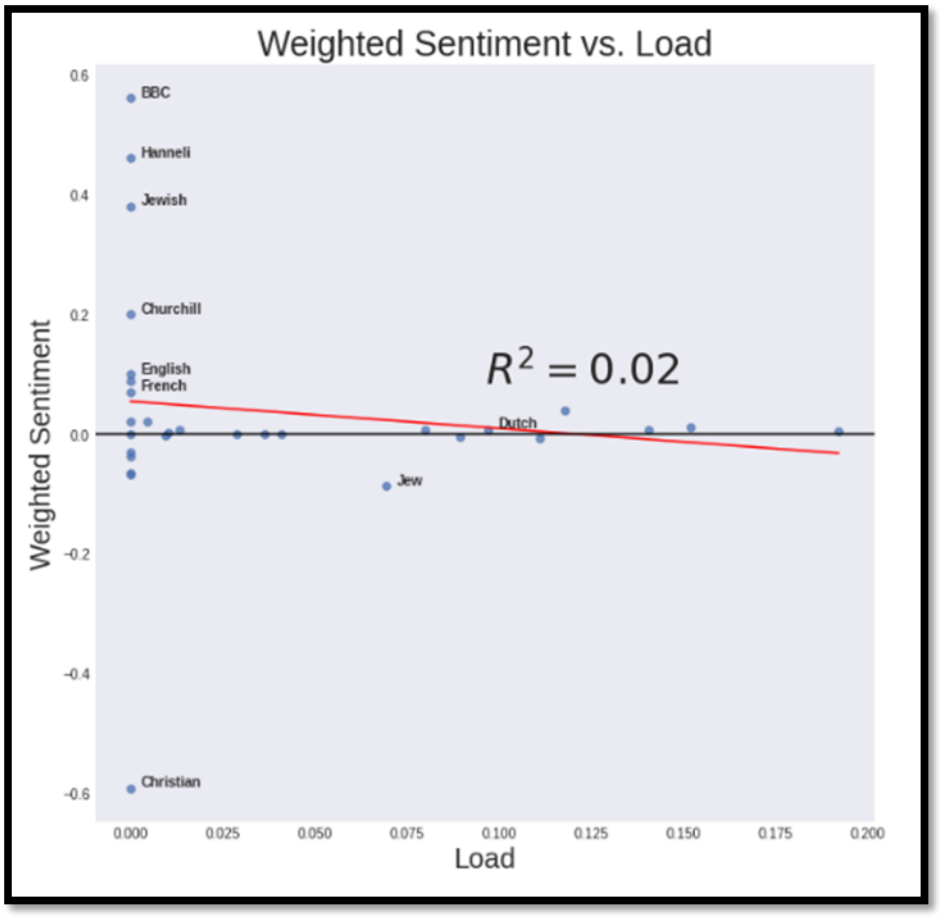

Figure L9: Relationship Between Weighted Sentiment & Load

Load, communicability, & closeness are overlaid for comparison in Figure L10. The overlapping circles of different centrality measures for a sentiment of -0.6 is once again for the term Christianity. It is evident now that higher centrality, for most terms, results in the averaging of sentiment such that there is a neutral result. The most significant exception to this rule is the term Jew which has been given the 'X' marker in Figure L10. Amongst terms of higher relative centrality, this term has relatively low sentiment. The implication here is that in more contexts than most, this term is the most associated with negativity despite the other terms that cooccurred with it. In this way, network theory again results in substantial overlap with the historical context of Anne's perspective during WWII.

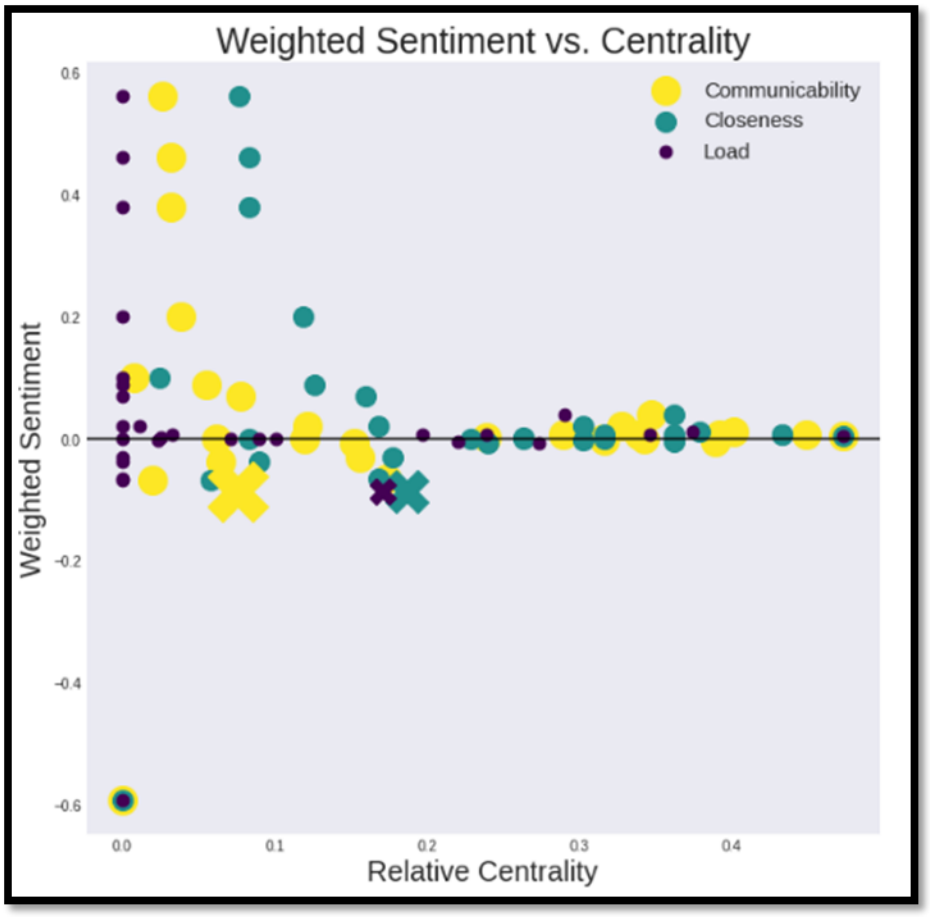

Figure K10: Weighted Sentiment vs. Relative Centrality Metrics

After a certain point in increasing centrality, well-connected terms have neutral to slightly positive sentiment. For the latter, the term surrounded by the most positive sentiment at the highest relative centrality across all three measures of centrality is Dutch. Dutch outranks other NORP's for every measure of centrality due to the duality it served between materials the helpers brought into the Annex as well as events external to the Annex. Since the helpers & Dutch often co-occurred in sentences, load centrality between these two types of Louvain communities reflect these shortest paths, most notably in 1944. The result was an increase in the Dutch term's nearest neighbors, thereby reducing its inverse distance to helpers in the Annex as opposed to other NORP's. German is a second NORP that also has a relatively high load, in part due to Anne's concern over the helpers getting caught. The term German shares similar connections with the same Louvain communities for this reason but lacks connections with the term Annex. In this way, the term German has a lower Katz centrality as well as a lower communicability centrality.

After comparing the relationship of centrality metrics with sentiment, there does not appear to be evidence Anne wrote terms of lower centrality rankings when she associated them with negative sentiment. Terms of lower centrality rankings extended to those with highly positive sentiment as well. For the residents of the Annex, Anne wrote about both the positives and negatives of these characters equally. For the NORP's focused on events outside the Annex, these terms exhibit low centrality rankings and, as such, are framed often only in specific contexts. Dutch, unlike most NORP's, has a variety of contexts that it was used in as it related to the inner workings of the Annex.

k.Network Statistical Metrics

The density of a graph is also an interesting network statistic to understand how well-connected each node is with each other. The density of this fully connected graph involving the terms for sentential co-occurrence was 0.257, meaning about 26% of all edges that could have existed for a complete graph were present (Barabási & Márton, 2017). This number does not factor in the weighting of edges. The weighted density is used to asses how much of the total graph is connected by edges relative to the max weight that exists between any two nodes. The weighted density was 0.00243. A low weighted density makes sense considering the disparity in the number of co-occurrences between terms required edge widths be displayed logarithmically.

The graph was not scale-free, meaning that the distribution of degrees does not follow a power-law. Figure M1 shows the distribution of node degree as a degree histogram. If this graph followed a power-law distribution, the number of nodes would primarily decrease with an increasing degree. Instead, there are a significant number of nodes with higher degrees more like that of a random network. The same is true for the weighted degree distribution of G, as shown in Figure M2.

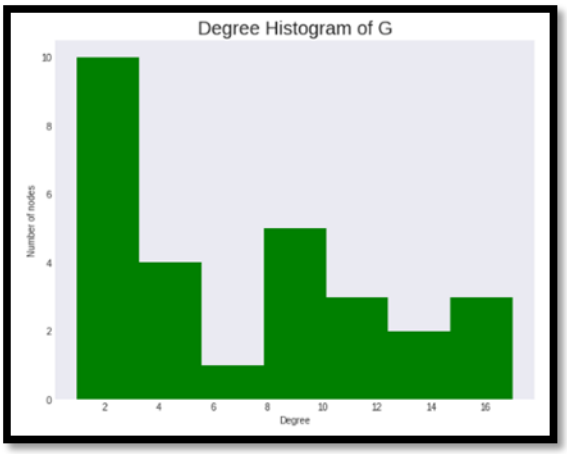

Figure M1: Degree Distribution of G

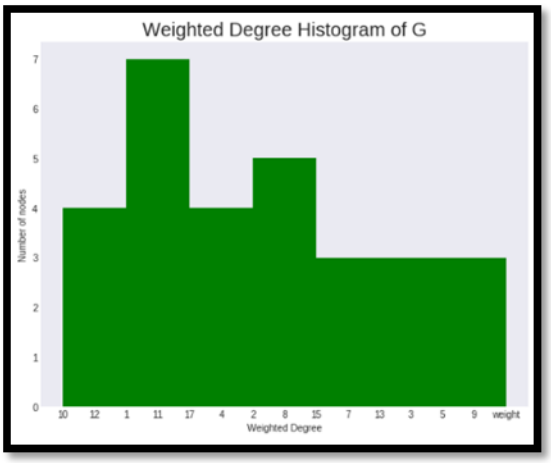

Figure M2: Weighted Degree Distribution of G

As a measure of how interconnected a graph is, network statistics typically refer to transitivity to measure the number of triangles in a graph or how close every node is to being part of a clique. The transitivity measure for this graph was 0.592, which implies this graph is neither fully interconnected nor relatively disconnected.

The clustering coefficient is another measure of how close a node is to being surrounded by cliques. Figure M3 shows the weighted clustering coefficients for this graph. Note how the tertiary nodes that were only connected to the graph with one edge, like French, have a coefficient of 0 since they have no surrounding clique or interconnectedness. Dussel & Boche have the highest clustering coefficients. The Boche node experiences this because that node has only two edges that are involved in a clique, all with relatively high weight. The reason for this is that the warehouse cat Boche primarily only co-occurred with the term Peter, who would sometimes play with the cat. The Dussel node, on the other hand, has a high degree of edges, and all collectively have a relatively high weight on average. The average weighted clustering of all nodes below is 0.059.

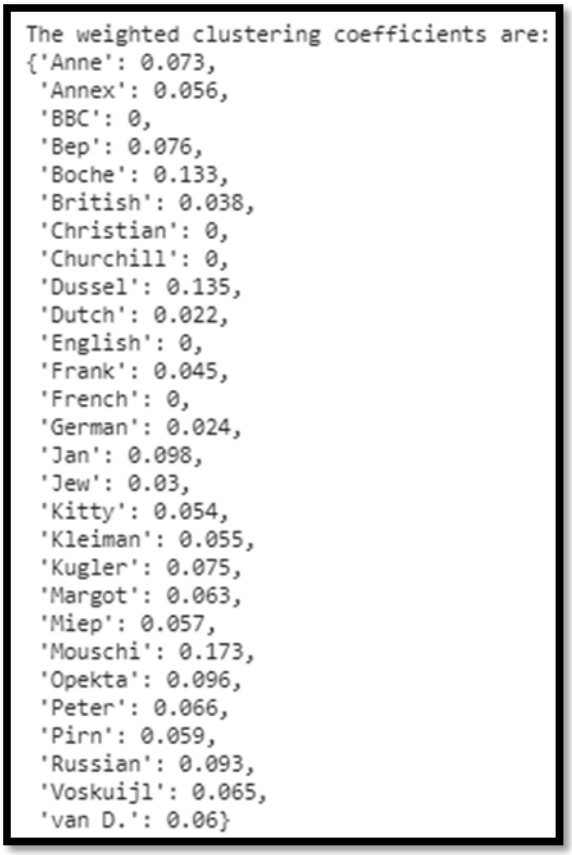

Figure M3: Weighted Clustering Coefficients by Node

Another way to assess the interconnectedness of a graph is to calculate its diameter. Diameter is calculated by first finding the shortest paths between each pair of vertices. Diameter is the longest of any of these shortest paths. The diameter of this graph was 4. For reference in terms of the shortest paths used to find this figure, the average shortest path was 2.08, far lower than the diameter. The weighted average shortest path was 4.22, where the shortest route is then the path between any two vertices with the minimum sum of edge weights along that path.

The last metric investigated for this graph was assortativity. Assortativity is the preference or likelihood for nodes to be attached to other like-nodes based upon a particular attribute. The degree assortativity of this graph, or the likelihood of nodes to be connected to

other like-degree nodes, was -0.15. A negative assortativity value implies disassortativity based upon that attribute. Nodes with higher degree were more likely for this graph to connect to nodes of lower degree and vice versa. The attribute assortativity coefficient using Louvain communities (rather than degree) is 0.33, showing intra-community nodes were more likely to have edges with other nodes in their respective community.

The mixing matrix of the graph in Figure M4 shows that most edges of the NLP tag Person and NORP are with other nodes of the same type. Product & ORG, however, mix with nodes of different types. Aside from edges from Person to the same type, as was seen in this project, Person most frequently connected to NORP followed by Product (Annex).

Mixing matrix of G by NLP tag:

	PERSON	NORP	PRODUCT	ORG
PERSON	124	11	10	3
NORP	11	20	1	0
PRODUCT	10	1	0	0
ORG	3	0	0	0

Figure M4: Mixing Matrix by NLP Tag

VII.Conclusions

The findings of this network analysis reveal the interplay of proper nouns & their relationships in Anne's diary as well as illustrate the overall graph structure brought into focus by data/network science. Research by Barabási et al. (2000) in the development of the Louvain method initially found it to be a superior community detection algorithm, and it was still considered to be one of the best community techniques for this research. Traag et al. (2019) introduced the Leiden algorithm as an improvement on the Louvain method, yet the results of the Leiden method only matched that of the Louvain method after finding the same community

membership. The Louvain method revealed the dynamic between the separate communities of the members of the Annex, the Annex's assistants, and the nationalities surrounding the Annex. Though Leiden & Louvain community detection methods were well-matched in measures of modularity and term membership, fluid community detection was found not to be a suitable method of community detection for this project.

Two reasons cited by Parés et al. (2017) to use fluid community detection is first to avoid "the creation of monster communities" and second to specify the number of communities in order "to find different levels of granularity." While it is true that modularity maximization algorithms like Leiden & Louvain may obtain communities that are too large, that was not found to be the case for this research. Since Parés et al. (2017) had k unknown communities with a fixed dataset, they were able to explore iterative values of k communities and compare them to those discovered by the fixed-community approach of the Louvain method. For this project, several aspects (most notably temporal) significantly affected the membership and modularity of both the fixed and dynamic community detection approaches. Too many variables in the discovery stages of how network theory applies to historical texts may have inhibited a fuller exploration of situations where fluid community detection may outperform Louvain & Leiden methods.

The same community detection methods were employed as they relate to time for this project. Min & Park (2019), in their analysis of Les Misérables, found that characters can exchange information over time that influences their community membership. The year 1944 found the highest assortativity for both NLP & the Louvain community detection algorithm. Moreover, the communities that the Louvain method found for 1944 directly correlated with

the text by primarily differentiating between Anne's close friends, NORP's, members of the Annex, and helpers of the Annex. One interest posited by this project for future research is if this is a regular occurrence across other historical texts. That is, if most diaries are written in such a way that the intra-community edges result in higher modularity over time. It is possible that this behavior only applies to autobiographical texts in which an author who recently went through a state of flux achieves a more steady-state situation.

Unlike many real-world networks, the overall graph of Anne Frank's diary was found not to be scale-free. This network statistic differed from the findings of Waumans et al. (2015) when an analysis of several popular novels was found to be scale-free. Partly, this may be due to size: *The Diary of a Young Girl* is only 50,000 words long, whereas series studied by Waumans et al. (2015) were often twenty times that at 1,000,000 words. How quickly diaries grow to be scale-free with the number of terms as well as even co-authors as variable inputs should be a focus of further research. Additionally, since scale-free networks exhibit preferential attachment, it may also be that Anne's confinement, as well as duress, prevented her from writing about regularly occurring relationships as they might be observed in nature. Subgraphs of communities that go from scale-free to not scale-free in-and-of-itself may be a sign of oppression in social networks. At the beginning of the diary, the extent of Anne's social sphere had already been reduced such that many proper nouns related discursively. Time may also be a critical limiting factor since members of such communities may have a limited duration to express themselves through different forms of media. Research studying the growth of single-author networks needs to be conducted to shed more light on this behavior. First, a network that is known to grow into one that is scale-free can have a power-law distribution to it fit like

that of a moving average temporally. Then, this metric for the fit of this distribution should be plotted against time and, separately, the rate of term growth. Once the threshold rate of growth or time needed for scale-free networks can be established, other networks from authors writing under different circumstances should be studied. If those networks meet the growth requirements to be scale-free but lack preferential attachment as seen by the power law, the circumstances surrounding each author should be noted.

Each centrality measure illustrates different roles each term took on throughout the diary. For example, Katz centrality directly correlated with the function Mr. Kleiman took upon himself in cheering up the residents of the Annex and making them feel looked after. Likewise, the substantial overlap was consistently found elsewhere between the definition of each centrality measure and the terms that ranked highest for that measure. Ranking terms, according to these measures, showed the overall most essential words to Anne in the diary. The highest-ranked terms included van D., Margot, Peter, & Dutch. These terms encapsulate Anne's frequent uses of her diary, including her documenting of interpersonal relations in the Annex, describing her sense of family, questioning her romantic interest, and forming her sense of identity. Though focusing on one measure of centrality may only reveal one aspect for the most well-connected member of a given Louvain community, the NetworkX library proved to have a vast number of different approaches to centrality that differed enough from each other to bring these more globally important terms to surface.

Benigni (2017) inspired the more humanitarian aspects of this research by identifying ISIS communities in social networks. ISIS community identification served as a template of how network science could be used as a preventative sign for humanitarian intervention. In the

study of Anne Frank's diary, nationality communities were found both by NLP as well as by the Louvain and Leiden methods. The sentiment within the community of nationalities was often strongly negative, indicating that measuring negative sentiment between NORP's might provide an early detection sign for political instability and prosecution.

The change in sentiment over time was also explored in this project, finding a very positive slope between British & German versus a very negative slope between British & Dutch. Anne's feelings grew the same for both pairs in that she felt the British were failing in its goal of liberating the Dutch. The positive slope between British & German was due to less warfare between the British & German forces when more fighting was needed to liberate Anne. Using almost any library for sentiment will detect less fighting over time as a positive change in sentiment. If humanitarian aid solutions are ever to utilize network science properly, negative & positive sentiment for different terms must be fully understood. In some cases, the reduction in negative sentiment will not be a positive outcome.

Sentential co-occurrence did not capture all relationships of terms in the text. This finding was discovered when looking at the change in sentiment over the diary entries in 1943. As Anne wrote more about individual relationships, terms would occur in nearby sentences but not co-occur sententially. Not discovering these relationships was an undesirable effect, mainly because the author was giving more information about a relationship, not less. Another facet not explored was the complicated relationship of sentiment via sentence proximity. Like the Katz measure of centrality, sentiment in neighboring sentences would then affect the sentiment in sentential co-occurrence. Here, to achieve a more nuanced look at sentiment score, term co-occurrence would act as nodes, edge width could be term proximity (same-

sentence being the highest), & the average edge width around each node would result in aggregate sentiment scores. Similar to how Katz centrality measures the nearby neighbor effect, nearest neighbors in sentences could influence the overall weighted sentiment via sentential proximity.

As the topic of network analysis pertains to text analysis, the overlap of the context of the diary and, separately, the findings of this analysis suggest that network analysis might be used for aggregating meaning from text documents. Identifying this overlap constitutes the primary goal of the exploratory work undertaken in this research through the lens of data/network science. Complex relationships appeared and reappeared over several different types of network analysis. These terms included Jew, Dutch, & French. Jew & French were observed in 1944 to be brokers between cliques that otherwise appeared to have fewer connections due to adults discouraging specific discussions about the war publicly. What was permitted were the declarations of Jewish perseverance & the study of language -which, given Anne's fondness of Peter, played a pivotal role between Louvain communities in 1944.

Dutch, in the context of the diary, represented what Anne long dreamed of in the Annex: freedom. As it affected this analysis, Anne longed for Dutch goods from the helpers, studied Dutch literature with Peter, and reflected on how the Dutch related to other NORP's in the war. For this reason, this term ranked consistently highest in measures of centrality versus the rest of the NORP's. Future research studying authors facing prosecution should not overlook the importance of using network theory to look at NORP's as it relates to the author's sense of identity. Additionally, sentiment between the terms an author identifies with may be illustrative in understanding both the author and the author's circumstances. Knowing Anne

identified with the terms Dutch and Jew was an essential aspect during this project in understanding sentiment around those terms, e.g., the negative sentiment between Jew & Christian as well as the negative sentiment between Dutch & British.

Finally, network analysis to detect authors that may be facing oppression or signs of duress should continue as a legitimate form of early, preventative detection. By comparing test cases of diaries and other types of print, specific aggregate patterns regarding sentiment summarization between Person & NORP might exist (or the assortativity by community attribute therein). At the time of writing this, few methods exist in accomplishing this task without having to gather several different libraries in Python after manually removing proper nouns that have been identified incorrectly. In contrast, the analysis of social media has been significantly simplified using comprehensive libraries, notably the Twitter API.

The existence of comprehensive libraries in Python for the network analysis of historical texts could add much-needed context to essential periods in human history. All the more necessary, their creation would not only standardize the analyses of texts like Anne Frank's diary but also allow the aggregation of these metrics across multiple texts from that period. With said aggregations, questions that arose about how the findings in this research correlate with the author & the author's situation could best be compared. One of the primary goals of this research was to correlate co-occurrent sentiment in such a way that the true feelings of the author would be known. An essential component for that goal would be in many ways met by additionally understanding how Anne's feelings significantly differed from others who had written texts during this period.

References

Anderson, J. M. (2007). *The Grammar of Names*. Oxford, United Kingdom: Oxford University Press.

Barabási, A.-L., Albert, R., & Jeong, H. (2000). Scale-free characteristics of random networks: the topology of the world-wide web. *Physica A: Statistical Mechanics and Its Applications*, 281(1-4), 69–77. doi: 10.1016/s0378-4371(00)00018-2

Barabási, A.-L., & Márton, P. (2017). *Network Science*. Cambridge, United Kingdom: Cambridge University Press.

Barnes, J., & Harary, F. (1983). Graph theory in network analysis. *Social Networks*, 5(2), 235–244. doi: 10.1016/0378-8733(83)90026-6

Benigni, M. C. (2017). Detection and Analysis of Online Extremist Communities. Institute for Software Research School of Computer Science Carnegie Mellon University, 1–143. Retrieved from http://reports-archive.adm.cs.cmu.edu/anon/anon/usr0/ftp/usr/ftp/isr2017/CMU-ISR-17-108.pdf

Benzi, M., & Klymko, C. (2013). Total communicability as a centrality measure. Journal of *Complex Networks*, 1(2), 124–149. doi: 10.1093/comnet/cnt007

Bird, S., Klein, E., & Loper, E. (2009). *Natural language processing with Python*. Bejing: OReilly.

Blazer, C. (1954). Front view of the house, Victor (Kraler) Kugler's office and business. [photograph]. Amsterdam, Netherlands: MAI.

Blondel, V. D., Guillaume, J.-L., Lambiotte, R., & Lefebvre, E. (2008). Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10). doi: 10.1088/1742-5468/2008/10/p10008

Cranmer, S. J., Desmarais, B. A., & Kirkland, J. H. (2012). Toward a Network Theory of Alliance Formation. *International Interactions*, 38(3), 295–324. doi: 10.1080/03050629.2012.677741

Freeman, L. C. (2004). The Development of Social Network Analysis. Empirical Press.

Gabbay, M. (2007). The effects of nonlinear interactions and network structure in small group opinion dynamics. *Physica A: Statistical Mechanics and Its Applications*, 378(1), 118–126. doi: 10.1016/j.physa.2006.11.051

Honnibal, M., & Johnson, M. (2015). An Improved Non-monotonic Transition System for Dependency Parsing. *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. doi: 10.18653/v1/d15-1162
Lancichinetti, A., & Fortunato, S. (2014). Community detection algorithms: A comparative analysis (vol 80, 056117, 2009). *Physical Review E, 89*(4), Physical Review E, 2014 Apr 7, Vol.89(4).

Liu, B. (2015). *Sentiment Analysis: Mining Options, Sentiments, and Emotions* (1st ed.). Cambridge University Press.

Love, B.C., & Sloman, S.A. (1995). Mutability and the determinants of conceptual transformability. In *Proceedings of the Seventeenth Annual Conference of the Cognitive Science Society* (pp. 654–659). Hillsdale, NJ: Erlbaum.

Manning, C. D., & Schutze, H. (2008). Foundations of statistical natural language processing. Cambridge, MA: MIT.

Min, S., & Park, J. (2019). Modeling narrative structure and dynamics with networks, sentiment analysis, and topic modeling. *PLOS One*, 14(12). doi: 10.1371/journal.pone.0226025

Newman, M. (2019). *Networks* (2nd ed.). Oxford: Oxford University Press.

Frank, O. M. (1997). *Anne frank: diary of a young girl*. New York, NY: Bantam Books, Inc.

Parés, F., Gasulla, D. G., Vilalta, A., Moreno, J., Ayguadé, E., Labarta, J., Suzumura, T. (2017). Fluid Communities: A Competitive, Scalable and Diverse Community Detection Algorithm. *Studies in Computational Intelligence Complex Networks & Their Applications* VI, 229–240. doi: 10.1007/978-3-319-72150-7_19

Perry, B. L., Pescosolido, B. A., & Borgatti, S. P. (2018). *Egocentric network analysis: foundations, methods, and models*. Cambridge, United Kingdom: Cambridge University Press.

Quirk, R., Greenbaum, S., Leech, G., & Svartvik, J. (1985). *A comprehensive grammar of the English language*. London: Longman.

van Wessel, C. & Ruys, F. (2013). Anne Frank's hiding place. [infographic]. Amsterdam, Netherlands. Retrieved from https://www.vizualism.nl/construction-kit-anne-franks-hiding/

Traag, V., Waltman, L., & Van Eck, N. (2019). From Louvain to Leiden: Guaranteeing well-connected communities. *Scientific Reports, 9*(1), 5233.

Wasserman, S., & Faust, K. (1994). *Social network analysis: methods and applications*. New York: Cambridge University Press.

Waumans, M. C., Nicodème, T., & Bersini, H. (2015). Topology Analysis of Social Networks Extracted from Literature. *PLOS One*, 10(6). doi: 10.1371/journal.pone.0126470

Appendix

```
#Source https://archive.org/stream/AnneFrankTheDiaryOfAYoungGirl_201606/Anne-Frank-The-Diary-Of-A-Young-Girl_djvu.txt
import re
import nltk
import math
import spacy
import random
import inflect
import datetime
import community
import leidenalg
import numpy as np
import igraph as ig
import pandas as pd
import networkx as nx
from pprint import pprint
from spacy import displacy
from matplotlib import colors
from operator import itemgetter
import matplotlib.pyplot as plt
from collections import Counter
from matplotlib.pyplot import cm
from itertools import permutations
from scipy.ndimage.filters import gaussian_filter1d
from nltk.sentiment.vader import SentimentIntensityAnalyzer
import pygraphviz
from networkx.drawing.nx_agraph import graphviz_layout
import en_core_web_lg
nlp = en_core_web_lg.load()
downloaded = drive.CreateFile({'id':"1bKPRap5G-ZO2FLrS8WjdpScA6O4tgKAL"})
downloaded.GetContentFile('TheDiaryOfAYoungGirl.txt')
#Regular expressions to look for when a new diary entry is beginning
```

1. Detecting Terms in Anne Frank's Diary

```
random.seed(1000)
r = '(?:SUNDAY|MONDAY|TUESDAY|WEDNESDAY|THURSDAY|FRIDAY|SATURDAY), (?:JANUARY|FEBRUARY|MARCH|APRIL|MAY|JUNE|JULY|A
UGUST|SEPTEMBER|OCTOBER|NOVEMBER|DECEMBER) \d{1,2}, 19\d{2}\s*\n'
date_indexes = []
#Open the file
with open("TheDiaryOfAYoungGirl.txt", encoding="utf8", errors='ignore') as f:
  lines = f.readlines()
  for i, line in enumerate(lines):
    # if '1943' in line:
    #   #Make a new diary entry for date
    if re.match(r, line):
        date_indexes.append((line.strip(), i))
  dia = {}
  #Build dictionary of the date indices and their entries
  for i in range(0, len(date_indexes) - 1):
    cur_idx = date_indexes[i][1] + 1
    next_idx = date_indexes[i + 1][1] - 1
    dia.update({date_indexes[i][0]: ''.join(lines[cur_idx:next_idx])})
#pprint(dia['FRIDAY, MARCH 17, 1944'])
i = 0
dd = []
dto_list = []
for key in dia.keys():
  td = key.split(',')
  td = td[1]+td[2]
  td = td.lstrip()
  dto = datetime.datetime.strptime(td, '%B %d %Y')
  dto_list.append(dto)
```

```
    if i == 0:
        firstd = dto
        dd.append(0)
    else:
        dd.append((dto-firstd).days)
    i = i + 1
#Consecutive day count from the first entry
print(dd)
dic = {}
i = 0
#Build a dictionary of the days with their respective text
for key, value in dia.items():
    dic.update({dd[i] : value.replace('\n','')})
    i = i +1
pprint(dic[0])
p = inflect.engine()
d_tags = {}

#Remove these parts of existing entities, e.g. Dearest Kitty vs. Kitty
def clean_up(X,ex_ls):
    clean_l = ['Dearest','Dear','kittle','\'s','\\','The','the','didn\'t','hasn\'t','doesn\'t','every','8,1943','gize,'
            ,'MAY 7,1944','wheeled','My','wasn\'t','COMMENT','14g','I\'m','couldn\'t','Great','Old','future','looked'
            ,'willi','de\'','d\'']
    for clean in clean_l:
        if (clean in X) and (X not in ex_ls):
            X = X.replace(clean,'')
            X = X.lstrip()
            X = X.rstrip()
    if p.singular_noun(X) and (X not in ex_ls):
        X = p.singular_noun(X)
    return X

#Reg expressions to detect nlp entities and reclassify them to standardize
def standardize(X,ex_ls):
    #Things I'd like to standardize to aggregate better
    rep = {'^anne':'Anne','^van d':'van D.','home guard':'Home Guard','^gies &':'Gies & Co.'
        ,'secret annex':'Annex','^bep':'Bep','^miep':'Miep','^margot':'Margot'
        ,'^peter':'Peter','hanneli':'Hanneli','hiller':'Hitler','^g.z. e. s.':'G.Z.',
        '^olga meyen- dorff' : 'Olga Meyen-dorff','^ursul':'Ursula','kitty':'Kitty','das schon':'das schon'}
    for key,value in rep.items():
        if (re.match(key,X.replace('"','').lower())) and (X not in ex_ls):
            X = rep[key]
    return X

#Make entity tags of nlp for each diary or, later, sentence entry
def make_entities(dic,keep_l):
    for key, value in dic.items():
        arr = []
        doc = nlp(value.replace('\n',''))
        #Keep these types of nlp entities
        #Typo/model error + german corrections
        drop_t = ['it','you\'re','he','his','her','ei ght','bd\'"dl','she','i\'m','there','who','wasn\'t','didn\'t','that','here','we\'ll'
            ,'let','what','i','son','he\'ll','re','wouldn\'t','y\'d','ridicu lou','lady','one','fed','i\'d','\'oh','you\'d'
            ,'s','shh','tranquthty','civthzed','now fifteen','hadn\'t','sun','\'i','she\'ll','man','my hand','saw'
            ,'\'remem ber','re\'re','you\'ve','you\'ll','shuffiing','he\'d','bye','nitely','much tidier','semidarknes'
            ,'you','te','mean-','conceited','my head','we\'d','i\'d lain','you\'ve','mothproofed'
            ,'\'that','famthar','noth','mothjr','m r','clodiesline','diat','togedier','algebra','fadier','entsetzlich'
            ,'nie zu ersetzen','nothina','\'that','beina','future','secredy','hand','writina','dealina','enouah','runnina'
            ,'unfortu','house','jun','\'he','brrr','i. boy','mally','dread','pst','chin','phy','iii','head'
            ,'cinema ater','sore throat','ketde','baker','seething','silent','tired','sister','lady','balli'
            ,'ding-dong','sohn','ron','mor','himmelhoch jauchzend','zu tode','der mann','jan gy','das liebe','das schone'
            ,'urn gotte','keg','eastern','du spritzt schon','quicksilver','espe','brea','miep, bep','gy','du bist doch eine'
            ,'ich mach\'','ruddy','kitty, bep','ifhe','pedes apostolorum','das schon','1','everYthing']

        #Things inflect library handles poorly or to exclude from touching
        ex_ls = ['Swiss','William Louis','Theseus','Zeus','Oedipus','Peleus','Orpheus','Hercules','Oasis','Maria Theresa','Myron'
```

```
            ,'Phidias','Annex','Trees','Broks','Dutch Sasas','Kuperus']

        for X in doc.ents:
            s1 = X.text
            if (X.label_ in keep_l):
                s1 = clean_up(s1,ex_ls)
                s1 = standardize(s1,ex_ls)
                if (s1.lower() not in drop_t) and (s1):
                    arr.append((s1, X.label_))
        d_tags[key] = arr
    return d_tags

keep_l = ['PERSON','NORP','PRODUCT','ORG']
d_tags = make_entities(dic,keep_l)
pprint(d_tags[0])
```

2. Dictionary of Attributes of Terms (NLP Entities)

```
chars = []
full_tags = []
tag_count_dic = {}
j = 0
for key, value in d_tags.items():
    i = 0
    tags = []
    tag_count_dic[dto_list[j]] = {}
    for elem in value:
        full_tags.append(value[i])
        chars.append(value[i][0])
        tags.append(value[i][1])
        i += 1
    for tag in set(tags):
        tag_count_dic[dto_list[j]][tag] = tags.count(tag)
    j += 1
tag_dic = {}
node_dic = {}
drop = ''
for tag in full_tags:
    if tag[0] not in tag_dic.keys():
        tag_dic[tag[0]] = [tag[1]]
    else:
        tag_dic[tag[0]].append(tag[1])

for key in tag_dic.keys():
  node_dic[key] = {}
  if key not in drop:
    node_dic[key]['tag'] =  max(set(tag_dic[key]), key = tag_dic[key].count)

node_dic
```

3. Time Series of Attributes

```
dft = pd.DataFrame(tag_count_dic)
dft = dft.fillna(0).transpose()
plt.subplots(figsize=(18,12))
columns = list(dft)
i = 0
cmap = cm.get_cmap('tab10')
for column in columns:
    y = gaussian_filter1d(dft[column],sigma=2.5)
    x = list(tag_count_dic.keys())
    plt.fill_between(x,y, color=cmap.colors[i], alpha=0.4)
    plt.plot(x,y,color=cmap.colors[i],label=column)
    i += 1

plt.xlabel('Date',size=40)
```

```
plt.ylabel('NLP Tag Count',size=40)
plt.title('Tag Count Over Time',size=60)
plt.ylim(ymin=0)
plt.xlim(xmax=max(x),xmin=min(x))
plt.legend(loc='upper right',prop={'size':14})
plt.tick_params(axis='both', which='major', labelsize=20)
plt.tick_params(axis='x', which='major',rotation=-45)
#plt.setp(plt.xaxis.get_majorticklabels(), rotation=-45 )
#lines = dft.plot.line(figsize=(18,12))
```

4. Diary Co-Occurrences

```
unique_chars = list(set(chars))
unique_full_tags = list(set(full_tags))
print(sorted(unique_chars))

df = pd.DataFrame(columns = unique_chars, index = unique_chars)
df[:] = int(0)

for key,value in d_tags.items():
    arr = []
    for elem in d_tags[key]:
        arr.append(elem[0])
    arr.append(elem[0])
    for char1 in unique_chars:
        for char2 in unique_chars:
            if char1 in arr and char2 in arr:
                df[char1][char2] += 1
                df[char2][char1] += 1
df.head()
```

5. Sentential Co-Occurrences

```
full_txt = ''
sent_dic = {}
sentences = []
i = 0
for z,v in dia.items():
    v = v.replace('\n','')
    full_txt += v
    dia[z] = v

def make_sentences(full_txt):
    doc = nlp(full_txt)
    sent_list = []
    for i, token in enumerate(doc.sents):
        for entity in unique_chars:
            if entity in token.text:
                sent_list.append(token.text)
                break
    return sent_list

from wordcloud import WordCloud

plt.figure(figsize=(12,6))

filtered_txt = ''
last_word = ''
for word in full_txt.split():
  if (word in unique_chars) and (word != last_word):
    filtered_txt += " " + word
    last_word = word

# Generate a word cloud image
wordcloud = WordCloud(background_color="black",width=800,height=400).generate(filtered_txt)
```

```
# Display the generated image:
# the matplotlib way:
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.title('Wordcloud of Anne Frank\'s Diary',size=40)
plt.show()

i = 0
nltk.download('vader_lexicon')
sid = SentimentIntensityAnalyzer()
overall_sent_dic = {}
for z,v in dia.items():
  l1 = make_sentences(v)
  for sentence in l1:
    overall_sent_dic[i] = {}
    overall_sent_dic[i]['sentence'] = sentence
    overall_sent_dic[i]['date'] = z
    i += 1
overall_sent_dic[0]

sentiments = []
sent_list = make_sentences(full_txt)

i = 0
sent_dic = {}
while i < len(sent_list):
    sent_dic[i] = sent_list[i]
    i += 1
sent_tags = make_entities(sent_dic,keep_l)

sent_tags[0]
```

6. Sentiment of Sentential Co-Occurrences

```
for key in sent_tags.keys():
    sentiment = sid.polarity_scores(sent_list[key])
    ent_list = []
    for tag in sent_tags[key]:
        ent_list.append(tag[0])
    sentiments.append((ent_list,sentiment))
sentiments[0:5]

unique_chars.append('Mother')
unique_chars.append('Father')
unique_chars.append('Grandma')
unique_chars.append('Mr. van Daan')
unique_chars.append('Mr. van D.')
unique_chars.append('Mrs. van Daan')
unique_chars.append('Mrs. van D.')

for z,v in overall_sent_dic.items():
  unique_list = []
  word_list = [i.lower().strip(',') for i in v['sentence'].split()]
  word_list = [i.split('\'')[0] for i in word_list]
  for char in unique_chars:
    for w in word_list:
      if (char.lower()==w):
        unique_list.append(char)
    if (' ' in char) and (char in v['sentence']):
      unique_list.append(char)
  for word in unique_list:
    for word2 in unique_list:
      if ' ' in word2:
        word3 = word2.split(' ')
        for word4 in word3:
          if (word == word4) and (word in unique_list):
```

```
            unique_list.remove(word)
    overall_sent_dic[z]['sentiment'] = sid.polarity_scores(v['sentence'])['compound']
    overall_sent_dic[z]['cooccurrences'] = list(set(unique_list))
overall_sent_dic[1]

df_sent = pd.DataFrame.from_dict(overall_sent_dic).transpose()
df_sent.index.name = 'SentID'
df_sent.head()

columns=['Date','Persons','Average Sentiment','Sentential Sentiment','Text']
df_sent2 = pd.DataFrame(columns=columns)

tup = []
text_list = []
dto,last_dto = 0,0
for idx, (_, value) in enumerate(df_sent.iterrows()):
    date = value.date
    td = date.split(',')
    td = td[1]+td[2]
    td = td.lstrip()
    dto = datetime.datetime.strptime(td, '%B %d %Y')
    if ((dto != last_dto) or (idx==len(df_sent)-1)) and len(tup):
      df_sent2.loc[last_idx] = pd.Series({'Date':last_dto, 'Persons':list(set(x for l in [i for (i,j) in tup] for x in l)), 'Average Sentiment':sum([i[1] for i in
tup])/len([i[1] for i in tup]), 'Sentential Sentiment':tup,'Text':text_list})
      text_list = []
      tup = []
    last_dto = dto
    last_idx = idx
    if len(value.cooccurrences) > 1:
      tup.append((value.cooccurrences,value.sentiment))
      text_list.append(value.sentence)
df_sent2

sentiments2 = []
for element in sentiments:
    ent_comb = list(permutations(element[0],2))
    for ent_tup in ent_comb:
      sentiments2.append((ent_tup,element[1]['compound']))
sentiments2[0:25]

#Build dictionary of sentiments observed per edge
sent_dic = {}
for element in sentiments2:
    if element[0] not in sent_dic:
        sent_dic[element[0]] = [element[1]]
    else:
        sent_dic[element[0]].append(element[1])

duplicates = []
for z in sent_dic:
  if ((z[1],z[0]) in sent_dic) and ((z[1],z[0]) not in duplicates):
    duplicates.append(z)

for duplicate in duplicates:
  del sent_dic[duplicate]

len(sent_dic)
```

7. Graph Plots

```
def make_edges(df):
    #Find edge values to determine max
    edge_values = []
    edge_list = {}
    for index, row in df.iterrows():
        i = 0
```

```
        for col in row:
            if col > 0 and (index != row.index[i]):
                edge_values.append(col)
            i += 1

    #Get max edge weight as percentage of that max edge value
    for index, row in df.iterrows():
        i = 0
        for col in row:
            #Remove edges with no weight, self-referential edges, and edges already added with inverse keys
            if col > 0 and (index != row.index[i]) and ((df.columns[i], index) not in list(edge_list.keys())):
                weight = float(col)/max(edge_values)
                edge_list[(index, df.columns[i])] = weight
            i += 1
    return edge_list

def drop_nodes(df,drop_list):
  df = df.drop(columns=drop_list)
  df = df.drop(drop_list)
  return df

def make_G(df,cooccurrence,filter,node_dic,sent_dic,sentiments2,ego='',drop=''):

  if drop:
    df2 = drop_nodes(df,[drop])
  else:
    df2 = df.copy()

  df3 = df2.copy()
  if filter:
    df3[df3 < filter] = 0

  edge_list = make_edges(df3)

  if cooccurrence == 'sentences':
    remove_edges = []
    for edge in edge_list:
      if (edge not in sent_dic) and ((edge[1],edge[0]) not in sent_dic):
        remove_edges.append(edge)
    for edge in remove_edges:
      del edge_list[edge]

  node_list = set([i for t in edge_list for i in t])

  G = nx.Graph()
  G.add_nodes_from(node_list)
  G.add_edges_from(edge_list)

  if ego:
    G = nx.ego_graph(G, ego)

  n_dic = {}
  e_dic = {}
  if cooccurrence=='diary':
    for node in G.nodes():
      n_dic[node] = {}
      n_dic[node]['size'] = df3[node].sum()/2
      n_dic[node]['tag'] = node_dic[node]['tag']
    for edge in G.edges():
      if edge in edge_list:
        e_dic[edge] = {}
        e_dic[edge]['width'] = edge_list[edge]
      else:
        e_dic[edge] = {}
        e_dic[edge]['width'] = edge_list[(edge[1],edge[0])]
```

```
 else:
   for node in G.nodes():
     count = 0
     n_dic[node] = {}
     for sent in sent_dic:
       if (node == sent[0]) or (node == sent[1]):
         count += 1
     n_dic[node]['size'] = count
     n_dic[node]['tag'] = node_dic[node]['tag']
   for edge in G.edges():
     if edge in sent_dic:
       e_dic[edge] = {}
       e_dic[edge]['sentiment'] = round(np.mean(sent_dic[edge]),2)
       e_dic[edge]['width'] = [i[0] for i in sentiments2].count(edge)
     elif (edge[1],edge[0]) in sent_dic:
       e_dic[edge] = {}
       e_dic[edge]['sentiment'] = round(np.mean(sent_dic[(edge[1],edge[0])]),2)
       e_dic[edge]['width'] = [i[0] for i in sentiments2].count((edge[1],edge[0]))

 return G,n_dic,e_dic
```

7a. Full Graph, Diary Co-Occurrent

```
plt.subplots(figsize=(20,20))

node_scalar = 0.7
edge_scalar = 100

G_everything,n_dic,e_dic = make_G(df,'diary',0,node_dic,sent_dic,sentiments2)

pos = graphviz_layout(G_everything)

nx.draw_networkx_edges(G_everything, pos, width=[e_dic[i]['width']*edge_scalar for i in G_everything.edges], edge_color='b', style='solid', alp
ha=0.05)
nx.draw_networkx_nodes(G_everything, pos, with_labels=False,node_list=n_dic, font_size = 18, font_weight = 'bold',node_size=[n_dic[i]['size']
*node_scalar for i in G_everything.nodes])
#nx.draw_networkx_labels(G_everything, pos, with_labels=True,node_list=n_dic, font_size = 12, font_weight = 'bold')

plt.title('Diary Co-Occurrence of All Terms',size=40)
```

7b. Frequent Diary Co-Occurrent Terms (Anne Removed)

```
import math
plt.subplots(figsize=(10,10))

node_scalar = 3
edge_scalar = 10

G,n_dic,e_dic = make_G(df,'diary',14,node_dic,sent_dic,sentiments2,'','Anne')

pos = graphviz_layout(G)

nx.draw_networkx_edges(G, pos, width=[math.log(e_dic[i]['width']+1)*edge_scalar for i in G.edges], edge_color='b', style='solid', alpha=0.8)
nx.draw_networkx_nodes(G, pos, node_list=n_dic, font_size = 12, font_weight = 'bold',node_size=[n_dic[i]['size']*node_scalar for i in G.nodes])
nx.draw_networkx_labels(G, pos, with_labels=True,node_list=n_dic, font_size = 12, font_weight = 'bold')

plt.title('Diary Co-Occurrence of Filtered Terms (Anne Removed)',size=20)
```

7c. Sentiment of Sentential Co-Occurrent Terms (Anne Removed)

```
import matplotlib

plt.subplots(figsize=(10,10))

node_scalar = 80
edge_scalar = 5
```

```
G,n_dic,e_dic = make_G(df,'sentences',8,node_dic,sent_dic,sentiments2,'','Anne')

pos = graphviz_layout(G,'neato')

sentiments = [e_dic[i]['sentiment'] for i in G.edges]

if min(sentiments) >= 0:
  min_sent = -1
else:
  min_sent = min(sentiments)

if max(sentiments) <= 0:
  max_sent = 1
else:
  max_sent = max(sentiments)
from matplotlib.pyplot import cm
cmap = cm.get_cmap('viridis_r')

norm = matplotlib.colors.DivergingNorm(vcenter=0, vmin=min_sent, vmax=max_sent)

sentiment_colors = [cmap(norm(i)) for i in sentiments]

sm = plt.cm.ScalarMappable(cmap=cmap, norm=plt.Normalize(vmin=min_sent, vmax=max_sent))
sm.set_array([])
cbar = plt.colorbar(sm) #, orientation='horizontal')
cbar.ax.set_ylabel('Sentiment',size=30,rotation=270,labelpad=20)

nx.draw(G, pos, with_labels=True,node_list=n_dic, font_size = 12, font_weight = 'bold',node_size=[n_dic[i]['size']*node_scalar for i in G.nodes],
width=[math.log(e_dic[i]['width']+1)*edge_scalar for i in G.edges],edge_color=sentiment_colors)
plt.title('Sentential Co-Occurrences of Filtered Terms\n(Anne Removed) w/ Sentiment (1944)',size=20)

def avg_sent(G,e_dic):
  avsen_dic = {}
  for node in G.nodes:
    avsen_dic[node] = {}
    sen_lis = []
    tot_width = []
    for edge in e_dic.keys():
      if (node in edge[0]) or (node in edge[1]):
        sen_lis.append(e_dic[edge]['sentiment']*e_dic[edge]['width'])
        tot_width.append(e_dic[edge]['width'])
    avsen_dic[node]['sentiment'] = sum(sen_lis)/len(sen_lis)
    avsen_dic[node]['weighted_sentiment'] = sum(sen_lis)/(len(sen_lis)*sum(tot_width))
  return avsen_dic

sen_dic1 = avg_sent(G,e_dic)

def bar_sent(sen_dic1):
  from matplotlib.pyplot import cm
  sen_dic2 = {}
  for node in sen_dic1.keys():
    sen_dic2[node] = sen_dic1[node]['weighted_sentiment']
  D = {k: v for k, v in sorted(sen_dic2.items(), key=lambda item: item[1])}

  sentiments = D.values()

  if min(sentiments) >= 0:
    min_sent = -1
  else:
    min_sent = min(sentiments)

  if max(sentiments) <= 0:
    max_sent = 1
  else:
    max_sent = max(sentiments)
```

```
 cmap = cm.get_cmap('viridis_r')

 norm = matplotlib.colors.DivergingNorm(vcenter=0, vmin=min_sent, vmax=max_sent)

 sentiment_colors = [cmap(norm(i)) for i in sentiments]

 fig, ax = plt.subplots()

 # Add some text for labels, title and custom x-axis tick labels, etc.
 ax.set_ylabel('Average Sentiment',size=18)
 ax.set_title('Weighted Sentiment by Node (1944)',size=24)
 ax.legend()

 fig.tight_layout()

 fig.set_figheight(10)
 fig.set_figwidth(8)

 #sm = plt.cm.ScalarMappable(cmap=cmap, norm=plt.Normalize(vmin=min_sent, vmax=max_sent))
 #sm.set_array([])
 #cbar = plt.colorbar(sm) #, orientation='horizontal')
 #cbar.ax.set_ylabel('Sentiment',size=30,rotation=270,labelpad=20)

 plt.bar(range(len(D)), list(D.values()), align='center',color=sentiment_colors)
 plt.xticks(range(len(D)), list(D.keys()))

 plt.xticks(rotation=90)
 plt.axhline(y=0, color='k', linestyle='-')

 plt.show()
bar_sent(sen_dic1)

pd.DataFrame.from_dict(sen_dic1).T

test_b = pd.DataFrame.from_dict(overall_sent_dic).T
test_b.head(30)

# for d in set(test_b['date']):
#   print(d)

test_b = test_b[["cooccurrences","date","sentiment","sentence"]]
test_b

from scipy.stats import linregress

def sent_slope(e_dic,overall_sent_dic):
  slope_dic = {}
  for edge in e_dic.keys():
    #slope_dic[edge] = {}
    days = []
    sen_data = []
    for k in overall_sent_dic.keys():
      if (edge[0] in overall_sent_dic[k]['cooccurrences']) and (edge[1] in overall_sent_dic[k]['cooccurrences']):
        days.append(k)
        sen_data.append(overall_sent_dic[k]['sentiment'])
    if days and sen_data:
      slope = round(linregress(days,sen_data).slope,4)
    else:
      slope = 0

    if math.isnan(slope):
      slope = 0

    slope_dic[edge] = slope*1000

  return slope_dic
```

```
# from matplotlib.pyplot import cm

# plt.subplots(figsize=(20,10))

# node_scalar = 90
# edge_scalar = 3

# pos = graphviz_layout(G,'dot')

# sent_slope_dic = sent_slope(e_dic,overall_sent_dic)

# sentiments = [sent_slope_dic[i] for i in G.edges]

# if min(sentiments) >= 0:
#   min_sent = -1
# else:
#   min_sent = min(sentiments)

# if max(sentiments) <= 0:
#   max_sent = 1
# else:
#   max_sent = max(sentiments)

# cmap = cm.get_cmap('viridis_r')

# norm = matplotlib.colors.DivergingNorm(vcenter=0, vmin=-1, vmax=1)

# sentiment_colors = [cmap(norm(i)) for i in sentiments]

# sm = plt.cm.ScalarMappable(cmap=cmap, norm=plt.Normalize(vmin=-1, vmax=1))
# sm.set_array([])
# cbar = plt.colorbar(sm) #, orientation='horizontal')
# cbar.ax.set_ylabel('Slope of Sentiment',size=30,rotation=270,labelpad=30)

# nx.draw(G, pos, with_labels=True,node_list=n_dic, font_size = 12, font_weight = 'bold',node_size=[n_dic[i]['size']*node_scalar for i in G.nodes
],width=[math.log(e_dic[i]['width']+1)*edge_scalar for i in G.edges],edge_color=sentiment_colors)
# plt.title('Sentential Co-Occurrences of Filtered Terms\n(Anne Removed) w/ Sentiment Slope',size=20)

# def avg_sent_slope(G,sent_slope_dic,e_dic):
#   avsen_dic = {}
#   for node in G.nodes:
#     avsen_dic[node] = {}
#     sen_lis = []
#     tot_width = []
#     for edge in G.edges:
#       if (node in edge[0]) or (node in edge[1]):
#         sen_lis.append(sent_slope_dic[edge]*e_dic[edge]['width'])
#         tot_width.append(e_dic[edge]['width'])
#     print(node,sen_lis)
#     avsen_dic[node]['sentiment'] = sum(sen_lis)/len(sen_lis)
#     avsen_dic[node]['weighted_sentiment'] = sum(sen_lis)/(len(sen_lis)*sum(tot_width))
#   return avsen_dic

# sen_dic1 = avg_sent_slope(G,sent_slope_dic,e_dic)

# def bar_sent(sen_dic1):
#   from matplotlib.pyplot import cm
#   sen_dic2 = {}
#   for node in sen_dic1.keys():
#     sen_dic2[node] = sen_dic1[node]['weighted_sentiment']
#   D = {k: v for k, v in sorted(sen_dic2.items(), key=lambda item: item[1])}

#   sentiments = D.values()

#   if min(sentiments) >= 0:
```

```
#    min_sent = -1
#   else:
#    min_sent = min(sentiments)

#   if max(sentiments) <= 0:
#    max_sent = 1
#   else:
#    max_sent = max(sentiments)

#   cmap = cm.get_cmap('viridis_r')

#   norm = matplotlib.colors.DivergingNorm(vcenter=0, vmin=min_sent, vmax=max_sent)

#   sentiment_colors = [cmap(norm(i)) for i in sentiments]

#   fig, ax = plt.subplots()

#   # Add some text for labels, title and custom x-axis tick labels, etc.
#   ax.set_ylabel('Average Sentiment',size=18)
#   ax.set_title('Weighted Sentiment by Node (1944)',size=24)
#   ax.legend()

#   fig.tight_layout()

#   fig.set_figheight(10)
#   fig.set_figwidth(8)

#   #sm = plt.cm.ScalarMappable(cmap=cmap, norm=plt.Normalize(vmin=min_sent, vmax=max_sent))
#   #sm.set_array([])
#   #cbar = plt.colorbar(sm) #, orientation='horizontal')
#   #cbar.ax.set_ylabel('Sentiment',size=30,rotation=270,labelpad=20)

#   plt.bar(range(len(D)), list(D.values()), align='center',color=sentiment_colors)
#   plt.xticks(range(len(D)), list(D.keys()))

#   plt.xticks(rotation=90)
#   plt.axhline(y=0, color='k', linestyle='-')

#   plt.show()
# bar_sent(sen_dic1)
```

7d. Clustering Functions for Graph Plots

```
nlp_misclassifieds = {'Miep':'PERSON','Moortje':'PERSON'}
for node in nlp_misclassifieds:
  if node in node_dic:
    node_dic[node]['tag'] = nlp_misclassifieds[node]

def make_clusters_async(G,clusters=5):
    from networkx.algorithms import community
    from matplotlib import colors
    import itertools
    communities = list(nx.community.asyn_fluidc(G, clusters))
    i = 0
    node = {}
    cmap = cm.get_cmap('Set3')
    node_colors = []
    for entity in sorted(G.nodes):
        for n, com in enumerate(list(communities)):
            if entity in com:
                color = colors.to_hex(cmap.colors[n])
        node_colors.append(color)
    return node_colors

test_nodes = []
tag_to_train = 'NORP'
```

```
for node in G.nodes:
  if node in node_dic.keys():
    if node_dic[node]['tag'] == tag_to_train:
      test_nodes.append(node)
test_nodes

tags_that_exist = []
for node in G.nodes:
  tags_that_exist.append(n_dic[node]['tag'])
tags_that_exist = set(tags_that_exist)
tags_that_exist

i = 0
z = 0
h = 0
clusters = len(tags_that_exist)
while (i != (len(test_nodes)-h)) or (min(color_l) < (4-h)):
  node_colors = make_clusters_async(G,clusters)
  j = 0
  color_dic = {}
  while j < len(n_dic):
      color_dic[list(n_dic)[j]] = node_colors[j]
      j += 1
  max_color = {}
  for tnode in test_nodes:
      count = 0
      for tnode2 in test_nodes:
          if tnode != tnode2:
              if color_dic[tnode] == color_dic[tnode2]:
                  count += 1
          max_color[tnode] = count
  i = max(max_color.values())
  color_l = []
  for color in set(node_colors):
    color_l.append(node_colors.count(color))
  if z == 1000:
    h += 1
    z = 0
  z += 1

  if h > 4:
    break

x = 0
color_comm_membership = {}
for value in set(color_dic.values()):
  cluster = []
  for node in n_dic:
    if color_dic[node] == value:
      cluster.append(node)
  color_comm_membership[x] = cluster
  x += 1
i

cmap = cm.get_cmap('Set3')
for i in color_comm_membership:
  for node in color_comm_membership[i]:
    n_dic[node]['fluid_color'] = cmap.colors[i]
n_dic

import math
def make_coordinates(num,coord_scale=9):
  angle = 2*math.pi/num#+random.uniform(0, 1)
  angle_list = []
  i = 0
  while i < num:
```

```
    angle_list.append(i*angle)
    i += 1
  coordinates = []
  coordinates.append((0,-1*coord_scale))
  j = 1
  while j < len(angle_list):
    coordinates.append((round(math.cos(angle_list[j]-math.pi/2)*coord_scale,2),round(math.sin(angle_list[j]-math.pi/2)*coord_scale,2)))
    j += 1
  return coordinates
make_coordinates(len(tags_that_exist))

def sort_position_list(pos,x_dic,x_list,nodes=n_dic):
  pos_list = []
  num_nodes = []
  i = 0
  for l in x_list:
    position_dic = {}
    for node in nodes:
      if x_dic[node]['tag'] == l:
        position_dic[node] = pos[node]
    if position_dic:
      pos_list.append(position_dic)

  def getNumKeys(dict1):
    return len(dict1.keys())
  pos_list=sorted(pos_list, key=getNumKeys)[::-1]
  return pos_list

pos_list = sort_position_list(pos,n_dic,tags_that_exist)
coordinates = make_coordinates(len(tags_that_exist))
def sort_coord(coordinates,pos_list):
  i=0
  y_list = []
  miny,maxy = (), ()
  coord_list = []
  tag_list = []
  tag_type = ''
  for coord in coordinates:
    tag_type = n_dic[list(pos_list[i].keys())[0]]['tag']
    y_list.append(coord[1])
    tag_list.append((tag_type,len(pos_list[i].keys())))
    i += 1

  def getKey(item):
    return item[1]
  tag_list=sorted(tag_list, key=getKey)[::-1]

  for coord in coordinates:
    if coord[1] == max(y_list):
      maxy = coord
    if coord[1] == min(y_list):
      miny = coord
  coord_list.append(miny)
  coord_list.append(maxy)
  for coord in coordinates:
    if (coord != miny) and (coord != maxy):
      coord_list.append(coord)
  return coord_list,tag_list

coord_list,tag_list = sort_coord(coordinates,pos_list)
coord_list

interim_tag = ['']*len(tags_that_exist)
def getIndexTuple(tup_list,tag):
  i = 0
  for tup in tup_list:
```

```
    index = i
    if tup[0] == tag:
      break
    i += 1
  return index

for tag in tags_that_exist:
  ind = getIndexTuple(tag_list,tag)
  interim_tag[ind]=tag

tags_that_exist = interim_tag
tags_that_exist

def circular_position(x_list,coord_list,pos,pos_list,x_dic):
  cir_pos_list = []
  #k = 0.8
  i = 0
  radius_scalar = 0.25 #2.5/coord_scale
  for l in x_list:
    G1 = nx.Graph()
    node_list = []
    for node in G.nodes:
      if x_dic[node]['tag'] == l:
        node_list.append(node)
    G1.add_nodes_from(node_list)
    #pos = nx.spring_layout(G1, k, iterations=17)
    pos = nx.circular_layout(G1,scale=len(G1.nodes)*radius_scalar)
    #print(pos)
    for key in pos.keys():
      pos[key] = [pos[key][0]+coord_list[i][0],pos[key][1]+coord_list[i][1]]
    cir_pos_list.append(pos)
    i += 1
  return cir_pos_list

pos2 = {}
cir_pos_list = circular_position(tags_that_exist,coord_list,pos,pos_list,n_dic)
i=0
while i < len(cir_pos_list):
  for key in cir_pos_list[i].keys():
    pos2[key] = cir_pos_list[i][key]
  i += 1
pos2

!pip install circle-fit
import circle_fit as cf

def show_circle_layout(cir_pos_list,min_radius=1):
  cmap = cm.get_cmap('Set3')
  circle_list = []
  xrange = []
  yrange = []
  rmax = 0
  fig, ax = plt.subplots(figsize=(10,10))
  ax.set_aspect("equal")
  i = 0
  for element in cir_pos_list:
    points = list(element.values())
    if len(points) > 1:
      xc,yc,r,_ = cf.least_squares_circle(points)
    else:
      xc,yc,r = pos2[list(element.keys())[0]][0],pos2[list(element.keys())[0]][1],min_radius
    #circle = plt.Circle((xc, yc), r, color='r')
    #fig, ax = plt.subplots() # note we must use plt.subplots, not plt.subplt
    # (or if you have an existing figure)
    #fig = plt.gcf()
    #ax = fig.gca()
```

```
    if r > rmax:
      rmax = r
    circle_list.append((xc,yc,r))
    xrange.append(xc-r)
    xrange.append(xc+r)
    yrange.append(yc-r)
    yrange.append(yc+r)
    c=plt.Circle((xc,yc),r,color=cmap.colors[i])
    ax.add_artist(c)
      # Show/save figure as desired.
  plt.xlim(min(xrange),max(xrange))
  plt.ylim(min(yrange),max(yrange))
  plt.show()
  return circle_list,rmax

  #fig.savefig('plotcircles.png')
circle_list,rmax = show_circle_layout(cir_pos_list)

tag_cir_dic = {}
i = 0
for tag in tags_that_exist:
  tag_cir_dic[tag] = circle_list[i]
  i += 1
tag_cir_dic

circle_list

from sympy.solvers import solve
from sympy import Symbol
def find_xydelta(circle_list,rmax,offset=2):
  i = 0
  ref_circle = ()
  xychange = []
  for circle in circle_list:
    r = circle[2]
    if (r == rmax) and (i == 0):
      ref_circle = circle
      i = 1
  i = 0
  for circle in circle_list:
    x_cir,y_cir,r = circle[0],circle[1],circle[2]
    x_ref,y_ref,r_ref = ref_circle[0],ref_circle[1],ref_circle[2]
    x_move = 0
    y_move = 0
    if (r == rmax) and (i == 0):
      i = 1
    #elif abs(round(y_cir/y_ref,3)) == 1:
    #  y_move = y_cir-y_ref + offset + r_ref
    elif round(x_cir/x_ref,3) == 1:
      if y_cir > y_ref:
        y_move = y_cir-y_ref-r_ref-offset
      elif y_ref > y_cir:
        y_move = -(y_ref-y_cir-r_ref-offset)
    elif round(y_cir/y_ref,3) == 1:
      if x_cir > x_ref:
        x_move = x_cir-x_ref-r_ref-offset
      elif x_ref > x_cir:
        x_move = -(x_ref-x_cir-r_ref-offset)
    else:
      m = (y_cir-y_ref)/(x_cir-x_ref)
      b = y_cir-m*x_cir
      x = Symbol('x')
      ref_answer = solve((x - x_ref)**2+(m*x+b-y_ref)**2-(r_ref+offset)**2, x)
      x_ref_ans = 0
      for a in ref_answer:
        if (x_ref < a < x_cir) or (x_ref > a > x_cir):
```

```
        x_ref_ans = a
      y_ref_ans = x_ref_ans*m+b
    answer = solve((x - x_cir)**2+(m*x+b-y_cir)**2-(r)**2, x)
    x_ans = 0
    for a in answer:
      if (x_ref < a < x_cir) or (x_ref > a > x_cir):
        x_ans = a
      y_ans = x_ans*m+b
    x_move = round(x_ans - x_ref_ans,2)
    y_move = round(y_ans - y_ref_ans,2)
   xychange.append((-x_move,-y_move))
 return xychange

xychange = find_xydelta(circle_list,rmax)
xychange

def centralize_circles(cir_pos_list,xychange):
 i = 0
 pos_list2 = []
 while i < len(cir_pos_list):
  for key in cir_pos_list[i].keys():
   new_node_list2 = {}
   new_node_list2[key] = [cir_pos_list[i][key][0]+xychange[i][0],cir_pos_list[i][key][1]+xychange[i][1]]
   pos_list2.append(new_node_list2)
  i += 1
 pos_list2

 final_pos_dic = {}
 i=0
 while i < len(pos_list2):
  for key in pos_list2[i].keys():
   final_pos_dic[key] = pos_list2[i][key]
  i += 1
 return final_pos_dic

final_pos_dic = centralize_circles(cir_pos_list,xychange)
final_pos_dic

def add_center_circles(xychange,circle_list,names):
 i = 0
 G2 = nx.Graph()
 pos_dic = {}
 center_dic = {}
 sizes = []
 r_dic = {}
 node_colors = []
 cmap = cm.get_cmap('Set3')

 while i < len(circle_list):
  name = names[i]
  x = circle_list[i][0]+xychange[i][0]
  y = circle_list[i][1]+xychange[i][1]
  r_dic[name] = circle_list[i][2]*100000
  pos_dic[name] = [x,y]
  center_dic[name] = circle_list[i][2]
  node_colors.append(cmap.colors[i])
  i += 1

 G2.add_nodes_from(center_dic)
 for node in G2.nodes:
  sizes.append(r_dic[node])

 return G2,pos_dic,sizes,node_colors

def overall_graph(node_dic,edge_dic,tag_cir_dic,node_scalar=10000,edge_scalar=20):
 cmap = cm.get_cmap('Set3')
```

```
 tag_node_dic = {}
 tag_edge_dic = {}
 tag_pos_dic = {}
 i = 0
 for tag in set(tag_cir_dic):
  node_num = 0
  tag_node_dic[tag] = {}
  size_l = []
  for node in node_dic:
   if node_dic[node]['tag'] == tag:
    node_num += 1
    size_l.append(node_dic[node]['size'])
  tag_node_dic[tag]['node_num'] = node_num
  tag_node_dic[tag]['color'] = cmap.colors[i]
  tag_pos_dic[tag] = (tag_cir_dic[tag][0],tag_cir_dic[tag][1])
  tag_node_dic[tag]['radius'] = tag_cir_dic[tag][2]
  tag_node_dic[tag]['avg_size'] = sum(size_l)/len(size_l)
  i += 1

 for tag in tag_cir_dic:
  for tag2 in tag_cir_dic:
   if (tag != tag2) and ((tag2,tag) not in tag_edge_dic.keys()):
    sentiment_l = []
    width_l = []
    for edge in edge_dic:
     if (((node_dic[edge[0]]['tag'] == tag) and (node_dic[edge[1]]['tag'] ==  tag2)) or ((node_dic[edge[1]]['tag'] == tag) and (node_dic[edge[0]]['t
ag'] ==  tag2))):
      sentiment_l.append(edge_dic[edge]['sentiment'])
      width_l.append(edge_dic[edge]['width'])
    if sentiment_l and width_l:
     tag_edge_dic[(tag,tag2)] = {}
     tag_edge_dic[(tag,tag2)]['width'] = round(sum(width_l) / len(width_l),2)
     tag_edge_dic[(tag,tag2)]['sentiment'] = round(sum(sentiment_l) / len(sentiment_l),2)
 #tag_edge_dic = {z: v for z, v in tag_edge_dic.items() if v}
 G_all = nx.Graph()
 G_all.add_nodes_from(tag_node_dic)
 G_all.add_edges_from(tag_edge_dic)

 for edge in G_all.edges:
  if edge not in tag_edge_dic:
   tag_edge_dic[edge] = {}
   tag_edge_dic[edge]['width'] = tag_edge_dic[(edge[1],edge[0])]['width']
   tag_edge_dic[edge]['sentiment'] = tag_edge_dic[(edge[1],edge[0])]['sentiment']
   del tag_edge_dic[(edge[1],edge[0])]

 return G_all,tag_node_dic,tag_edge_dic,tag_pos_dic
G_all,tag_node_dic,tag_edge_dic,tag_pos_dic = overall_graph(n_dic,e_dic,tag_cir_dic)
```

7e. Overall Clustering of Sentential Co-Occurences by Term Attribute

```
def get_att_assort(G,n_dic,att='tag'):
 Gf=nx.Graph()
 i = 0
 att_list = []
 for node in G.nodes:
  att_list.append(n_dic[node][att])
 att_list =  set(att_list)
 for x in att_list:
  node_list = []
  for node in G.nodes:
   if x == n_dic[node][att]:
    node_list.append(node)
  Gf.add_nodes_from(node_list,att=n_dic[node_list[0]][att])
```

```
    i += 1
  Gf.add_edges_from(G.edges)
  return round(nx.attribute_assortativity_coefficient(Gf,'att'),2)

import matplotlib
plt.figure(figsize=(15,15))

edge_scalar = 20
node_scalar = 30000

sentiments = [e_dic[i]['sentiment'] for i in e_dic]
node_size = [tag_node_dic[i]['radius']*node_scalar for i in G_all.nodes]
colors = [tag_node_dic[i]['color'] for i in G_all.nodes]

tag_sentiment = [tag_edge_dic[i]['sentiment'] for i in G_all.edges]
widths = [math.log(tag_edge_dic[i]['width']+1)*edge_scalar for i in G_all.edges]

cmap = plt.cm.viridis_r
cmaplist = [cmap(i) for i in range(cmap.N)]
cmaplist[0] = (1.0,1.0,1.0,1.0)
cmap = matplotlib.colors.LinearSegmentedColormap.from_list('mcm',cmaplist, cmap.N)

norm = matplotlib.colors.DivergingNorm(vcenter=0, vmin=min_sent, vmax=max_sent)

sentiment_colors = [cmap(norm(i)) for i in tag_sentiment]

nx.draw(G_all,tag_pos_dic, with_labels=True, font_size = 18, font_weight = 'bold',alpha=1,node_size=node_size,node_color=colors)

nx.draw_networkx_edges(G_all,tag_pos_dic,edgelist=G_all.edges,width=widths,edge_color=sentiment_colors, alpha=1)

plt.title('Sentential Co-Occurrence by\nNLP Attribute Summarization',size=40)
sm = plt.cm.ScalarMappable(cmap=cmap, norm=plt.Normalize(vmin=min_sent, vmax=max_sent))
sm.set_array([])
cbar = plt.colorbar(sm) #, orientation='horizontal')
cbar.ax.set_ylabel('Sentiment',size=30,rotation=270,labelpad=20)
plt.xlim(-12.5,12.5)
plt.ylim(-15.2,13.7)
```

7f. Sentential Co-Occurence Clustering by Term Attribute

```
import matplotlib as mpl
from mpl_toolkits.axes_grid1 import AxesGrid

plt.figure(figsize=(20,20))

edge_scalar = 4
node_scalar = 200

sentiments = [e_dic[i]['sentiment'] for i in G.edges]
node_size = [n_dic[i]['size']*node_scalar for i in G.nodes]
colors = [n_dic[i]['fluid_color'] for i in G.nodes]

sentiment = [e_dic[i]['sentiment'] for i in G.edges]
widths = [math.log(e_dic[i]['width']+1)*edge_scalar for i in G.edges]

cmap = plt.cm.viridis_r
cmaplist = [cmap(i) for i in range(cmap.N)]
cmaplist[0] = (1.0,1.0,1.0,1.0)
cmap = matplotlib.colors.LinearSegmentedColormap.from_list('mcm',cmaplist, cmap.N)

norm = matplotlib.colors.DivergingNorm(vcenter=0, vmin=min_sent*1.01, vmax=max_sent*1.01)

sentiment_colors = [cmap(norm(i)) for i in sentiment]

nx.draw(G, final_pos_dic, with_labels=True, font_size = 18, font_weight = 'bold',node_size = [n_dic[i]['size']*node_scalar for i in n_dic], node_c
olor = colors, width = widths)
```

```
nx.draw_networkx_edges(G, final_pos_dic,edgelist=G.edges,width=[math.log(e_dic[i]['width']+1)*edge_scalar for i in G.edges], alpha=1, edge_
color=sentiment_colors)

sm = plt.cm.ScalarMappable(cmap=cmap, norm=plt.Normalize(vmin=min_sent, vmax=max_sent))
sm.set_array([])
cbar = plt.colorbar(sm)
cbar.ax.set_ylabel('Sentiment',size=30,rotation=270,labelpad=20)

plt.title('Sentential Co-
Occurrences by\nFluid Community (Color), Assort = {}\n NLP Object (Cluster), Assort = {}'.format(get_att_assort(G,n_dic,att='fluid_color'),get_at
t_assort(G,n_dic)),size=40)

import community
import operator

partition_lo = community.best_partition(G)

kk=len(set(partition_lo.values()))
modularity_lo=community.modularity(partition_lo, G, weight='weight')
print('The graph has', kk, 'Louvain communities and modularity coefficient equal to', modularity_lo)

cml=[]
for j in range(kk):
    cj=[]
    for k,v in partition_lo.items():
        if v==j:
            cj.append(k)
    cml.append(cj)
cmd={}
for j in range(kk):
    cmd[tuple(cml[j])]=len(cml[j])
cmd
ocmd=sorted(cmd.items(), key=operator.itemgetter(1), reverse=True)
eocmd=enumerate(ocmd)
commsd_lo={}
louvain_comm_membership={}
for i in eocmd:
    print('Community', i[0], 'with', i[1][1], 'nodes:', list(i[1][0]))
    commsd_lo[i[0]]=i[1][1]
    ts=list(i[1][0])
    louvain_comm_membership[i[0]]=ts

g = ig.Graph(directed=False)
g.add_vertices(list(G.nodes))
g.add_edges(G.edges)

partition_le = leidenalg.find_partition(g, leidenalg.ModularityVertexPartition);

partition_nodes = []
for part_list in partition_le:
  part = [list(G.nodes())[i] for i in part_list]
  partition_nodes.append(part)

cml=list(partition_nodes)
leiden_partition={}
for i,j in list(enumerate(cml)):
  for jj in j:
    leiden_partition[jj]=i
modularity_le=community.modularity(leiden_partition, G, weight='weight')
print('The graph has', kk, 'Leiden communities and modularity coefficient equal to', modularity_le)

cmd={}
kk=len(cml)
for j in range(kk):
  cmd[tuple(cml[j])]=len(cml[j])
```

```
cmd

ocmd=sorted(cmd.items(), key=operator.itemgetter(1), reverse=True)
eocmd=enumerate(ocmd)
commsd_le={}
leiden_comm_membership={}
for i in eocmd:
   print('Community', i[0], 'with', i[1][1], 'nodes:', list(i[1][0]))
   commsd_le[i[0]]=i[1][1]
   ts=list(i[1][0])
   leiden_comm_membership[i[0]]=ts

import matplotlib as mpl
from mpl_toolkits.axes_grid1 import AxesGrid

for z,v in louvain_comm_membership.items():
 for node in v:
   n_dic[node]['louvain_partition'] = 'Louvain_Comm_' + str(z)

plt.figure(figsize=(20,20))

cmap = cm.get_cmap('Set3')

edge_scalar = 3
node_scalar = 300

communities = list(set([n_dic[i]['louvain_partition'] for i in G.nodes]))
sentiments = [e_dic[i]['sentiment'] for i in G.edges]
node_size = [n_dic[i]['size']*node_scalar for i in G.nodes]
colors = [cmap.colors[communities.index(n_dic[i]['louvain_partition'])] for i in G.nodes]

sentiment = [e_dic[i]['sentiment'] for i in G.edges]
widths = [math.log(e_dic[i]['width']+1)*edge_scalar for i in G.edges]

cmap = cm.get_cmap('viridis_r')

norm = matplotlib.colors.DivergingNorm(vcenter=0, vmin=min_sent*1.01, vmax=max_sent*1.01)

#G2,pos,sizes_G2,node_colors_G2 = add_center_circles(xychange,circle_list,tags_that_exist)

sentiment_colors = [cmap(norm(i)) for i in sentiment]

nx.draw(G, final_pos_dic, with_labels=True, font_size = 18, font_weight = 'bold',node_size = [n_dic[i]['size']*node_scalar for i in G.nodes], node
_color = colors, width = widths)

nx.draw_networkx_edges(G, final_pos_dic,edgelist=G.edges,width=[math.log(e_dic[i]['width']+1)*edge_scalar for i in e_dic], alpha=1, edge_col
or=sentiment_colors)

sm = plt.cm.ScalarMappable(cmap=cmap, norm=plt.Normalize(vmin=min_sent, vmax=max_sent))
sm.set_array([])
cbar = plt.colorbar(sm)
cbar.ax.set_ylabel('Sentiment',size=30,rotation=270,labelpad=20)

plt.title('Sentential Co-
Occurrences by\nLouvain Community (Color), Assort = {}\n NLP Object (Cluster), Assort = {}'.format(get_att_assort(G,n_dic,'louvain_partition'),g
et_att_assort(G,n_dic)),size=40)
```

7g. Overall Clustering of Sentential Co-Occurences by Community

```
louvain_colors = []

cmap = cm.get_cmap('Set3')
```

```
for node in G.nodes:
 i = 0
 while i < len(louvain_comm_membership):
  if node in louvain_comm_membership[i]:
   n_dic[node]['louvain_color'] = cmap.colors[i]
  i += 1

k = 1.3
pos_part2 = nx.spring_layout(G,k)
pos_part2

node_colors = louvain_colors
coord_part2 = make_coordinates(len(louvain_comm_membership))
colors_that_exist = set(n_dic[i]['louvain_color'] for i in n_dic)
colors_that_exist

color_pos = []
for color in colors_that_exist:
 node_list2 = {}
 for node in G.nodes:
  if n_dic[node]['louvain_color'] == color:
   node_list2[node] = pos_part2[node]
 color_pos.append(node_list2)
color_pos

def circular_color_position(x_list,coord_list,pos,pos_list,n_dic):
 cir_pos_list = []
 #k = 0.8
 i = 0
 radius_scalar = 0.25 #2.5/coord_scale
 for l in x_list:
  G1 = nx.Graph()
  new_node_list = []
  for node in n_dic:
   if n_dic[node]['louvain_color'] == l:
    new_node_list.append(node)
  G1.add_nodes_from(new_node_list)
  #pos = nx.spring_layout(G1, k, iterations=17)
  pos = nx.circular_layout(G1,scale=len(G1.nodes)*radius_scalar)
  #print(pos)
  for key in pos.keys():
   pos[key] = [pos[key][0]+coord_list[i][0],pos[key][1]+coord_list[i][1]]
  cir_pos_list.append(pos)
  i += 1
 return cir_pos_list

cir_pos_list2 = circular_color_position(colors_that_exist,coord_part2,pos_part2,color_pos,n_dic)
circle_list2,rmax2 = show_circle_layout(cir_pos_list2,0.01)
xychange2 = find_xydelta(circle_list2,rmax2,3)
final_pos_dic2 = centralize_circles(cir_pos_list2,xychange2)

def color_by_tag(tags_that_exist,node_dic):
 i = 0
 cmap = cm.get_cmap('Set3')
 node_colors2 = []
 for node in node_dic:
  i = 0
  for tag in tags_that_exist:
   if node_dic[node]['tag'] == tag:
    node_colors2.append(cmap.colors[i])
   i += 1
 return node_colors2
node_colors2 = color_by_tag(tags_that_exist,n_dic)
node_colors2

louvain_pos = {}
```

```
i = 0
for v in sorted(set(louvain_comm_membership)):
  key = 'Louvain_Comm_'+ str(v)
  louvain_pos[key] = circle_list2[i]
  i += 1
louvain_pos

def overall_louvain_graph(node_dic,edge_dic,louvain_pos,node_scalar=10000,edge_scalar=20):
  cmap = cm.get_cmap('Set3')
  tag_node_dic = {}
  tag_edge_dic = {}
  tag_pos_dic = {}
  i = 0
  for tag in louvain_pos:
    node_num = 0
    tag_node_dic[tag] = {}
    size_l = []
    for node in node_dic:
      if node_dic[node]['louvain_partition'] == tag:
        node_num += 1
    tag_node_dic[tag]['node_num'] = node_num
    tag_node_dic[tag]['color'] = cmap.colors[i]
    tag_pos_dic[tag] = (louvain_pos[tag][0],louvain_pos[tag][1])
    tag_node_dic[tag]['radius'] = louvain_pos[tag][2]
    i += 1

  for tag in louvain_pos:
    for tag2 in louvain_pos:
      if (tag != tag2) and ((tag2,tag) not in tag_edge_dic.keys()):
        sentiment_l = []
        width_l = []
        for edge in edge_dic:
          if (((node_dic[edge[0]]['louvain_partition'] == tag) and (node_dic[edge[1]]['louvain_partition'] ==  tag2)) or ((node_dic[edge[1]]['louvain_p
artition'] == tag) and (node_dic[edge[0]]['louvain_partition'] ==  tag2))):
            sentiment_l.append(edge_dic[edge]['sentiment'])
            width_l.append(edge_dic[edge]['width'])
        if sentiment_l and width_l:
          tag_edge_dic[(tag,tag2)] = {}
          tag_edge_dic[(tag,tag2)]['width'] = round(sum(width_l) / len(width_l),2)
          tag_edge_dic[(tag,tag2)]['sentiment'] = round(sum(sentiment_l) / len(sentiment_l),2)
  #tag_edge_dic = {z: v for z, v in tag_edge_dic.items() if v}
  G_all = nx.Graph()
  G_all.add_nodes_from(tag_node_dic)
  G_all.add_edges_from(tag_edge_dic)

  for edge in G_all.edges:
    if edge not in tag_edge_dic:
      tag_edge_dic[edge] = {}
      tag_edge_dic[edge]['width'] = tag_edge_dic[(edge[1],edge[0])]['width']
      tag_edge_dic[edge]['sentiment'] = tag_edge_dic[(edge[1],edge[0])]['sentiment']
      del tag_edge_dic[(edge[1],edge[0])]

  return G_all,tag_node_dic,tag_edge_dic,tag_pos_dic
G_all,tag_node_dic,tag_edge_dic,tag_pos_dic = overall_louvain_graph(n_dic,e_dic,louvain_pos)

from matplotlib.pyplot import cm
plt.figure(figsize=(15,15))

edge_scalar = 14
node_scalar = 20000

plt.style.use('seaborn-deep')

sentiments = [tag_edge_dic[i]['sentiment'] for i in G_all.edges]
node_size = [tag_node_dic[i]['radius']*node_scalar for i in G_all.nodes]
colors = [tag_node_dic[i]['color'] for i in G_all.nodes]
```

```
tag_sentiment = [tag_edge_dic[i]['sentiment'] for i in G_all.edges]
widths = [math.log(tag_edge_dic[i]['width']+1)*edge_scalar for i in G_all.edges]

cmap = cm.get_cmap('viridis_r')

norm = matplotlib.colors.DivergingNorm(vcenter=0, vmin=min_sent*1.01, vmax=max_sent*1.01)

sentiment_colors = [cmap(norm(i)) for i in tag_sentiment]

nx.draw(G_all,tag_pos_dic, with_labels=True, font_size = 18, font_weight = 'bold',alpha=1,node_size=node_size,node_color=colors)

nx.draw_networkx_edges(G_all,tag_pos_dic,edgelist=G_all.edges,width=widths,edge_color=sentiment_colors, alpha=1)

sm = plt.cm.ScalarMappable(cmap=cmap, norm=plt.Normalize(vmin=min_sent, vmax=max_sent))
sm.set_array([])
cbar = plt.colorbar(sm)
cbar.ax.set_ylabel('Sentiment',size=30,rotation=270,labelpad=20)

plt.title('Sentential Co-Occurrence by\nLouvain Community Summarization (1944)',size=40)
plt.xlim(-11.5,12.5)
plt.ylim(-15.2,13.7)

for z,v in louvain_comm_membership.items():
  print( 'Louvain Community ' + str(z) + ' contains ' + str(v))

for node in G.nodes:
 print(node,n_dic[node])
```

7h. Sentential Co-Occurence Clustering by Community

```
import matplotlib as mpl
from mpl_toolkits.axes_grid1 import AxesGrid
from matplotlib.pyplot import cm

#G,n_dic,e_dic = make_G(df,'sentences',8,node_dic,sent_dic,sentiments2,'','Anne')

plt.figure(figsize=(20,15))

k = 0.8
edge_scalar = 2.5
node_scalar = 200

cmap = cm.get_cmap('viridis_r')
cmap2 = cm.get_cmap('Set3')

sm = plt.cm.ScalarMappable(cmap=cmap, norm=plt.Normalize(vmin=min_sent, vmax=max_sent))
sm.set_array([])
cbar = plt.colorbar(sm)
cbar.ax.set_ylabel('Sentiment',size=30,rotation=270,labelpad=20)

widths = [math.log(e_dic[i]['width']+1)*edge_scalar for i in G.edges]
sentiments = [e_dic[i]['sentiment'] for i in G.edges]
sizes = [n_dic[i]['size']*node_scalar for i in G.nodes]
node_color = [cmap2.colors[tags_that_exist.index(n_dic[i]['tag'])] for i in G.nodes]

nx.draw(G, final_pos_dic2, with_labels=True, font_size = 18, font_weight = 'bold',node_size = sizes, node_color = node_color, width = widths)

nx.draw_networkx_edges(G, final_pos_dic2,edgelist=G.edges,width=widths, alpha=1, edge_color=sentiments, edge_cmap=cmap)

plt.title('Sentential Co-
Occurrences in 1944 by\nNLP Object (Color), Assort = {}\n Louvain Community (Cluster), Assort = {}'.format(get_att_assort(G,n_dic),get_att_ass
ort(G,n_dic,'louvain_partition')),size=40)
```

7i. Egocentric Graph of Diary/Sentential Co-Occurent Terms (Annex/Kitty/Anne)

```
plt.subplots(figsize=(15,15))

node_scalar = 250
edge_scalar = 3

ego_name = 'Kitty'

Gg,n_dic,e_dic = make_G(df,'sentences',0,node_dic,sent_dic,sentiments2,ego_name,'')

#pos = graphviz_layout(G)

sentiments = [e_dic[i]['sentiment'] for i in Gg.edges]

if min(sentiments) >= 0:
  min_sent = -1
else:
  min_sent = min(sentiments)

if max(sentiments) <= 0:
  max_sent = 1
else:
  max_sent = max(sentiments)

cmap = cm.get_cmap('viridis_r')
cmap2 = cm.get_cmap('Set3')

norm = matplotlib.colors.DivergingNorm(vcenter=0, vmin=min_sent*1.01, vmax=max_sent*1.01)

sentiment_colors = [cmap(norm(i)) for i in sentiments]

color = []
i = 0
for node in Gg.nodes:
  if node == ego_name:
    color.append(cmap2.colors[i])
  else:
    color.append(cmap2.colors[i+1])

sm = plt.cm.ScalarMappable(cmap=cmap, norm=plt.Normalize(vmin=min_sent, vmax=max_sent))
sm.set_array([])
cbar = plt.colorbar(sm)
cbar.ax.set_ylabel('Sentiment',size=30,rotation=270,labelpad=20)

node_size = []
for node in Gg.nodes:
  if node != ego_name:
    node_size.append(n_dic[node]['size']*node_scalar)
  else:
    node_size.append(n_dic[node]['size']*node_scalar*1.2)

# node_and_degree = Gg.degree()
# (largest_hub, degree) = sorted(node_and_degree, key=itemgetter(1))[-1]
largest_hub=ego_name
# Create ego graph of main hub
hub_ego = nx.ego_graph(Gg, largest_hub)
# Draw graphu
pos = nx.spring_layout(hub_ego)
#hub_ego.remove_node(ego_name)
nx.draw(hub_ego, pos, node_color=cmap2.colors[4], node_size=node_size, with_labels=True,font_size=12,font_weight='bold',width=[math.log
(e_dic[i]['width']+1)*edge_scalar for i in Gg.edges],edge_color=sentiment_colors)
# Draw ego as large and red
```

```
nx.draw_networkx_nodes(hub_ego.remove_node(ego_name), pos, nodelist=[largest_hub], node_size=n_dic[ego_name]['size']*node_scalar, n
ode_color=cmap2.colors[3], with_labels=True,font_size=12,font_weight='bold')

plt.title('Egocentric Sentential Co-Occurrence of {}'.format(ego_name),size=40)

plt.subplots(figsize=(15,15))

node_scalar = 250
edge_scalar = 3

ego_name = 'Anne'

Gg,n_dic,e_dic = make_G(df,'sentences',6,node_dic,sent_dic,sentiments2,ego_name,'')

#pos = graphviz_layout(G)

sentiments = [e_dic[i]['sentiment'] for i in Gg.edges]

if min(sentiments) >= 0:
  min_sent = -1
else:
  min_sent = min(sentiments)

if max(sentiments) <= 0:
  max_sent = 1
else:
  max_sent = max(sentiments)

cmap = cm.get_cmap('viridis_r')
cmap2 = cm.get_cmap('Set3')

norm = matplotlib.colors.DivergingNorm(vcenter=0, vmin=min_sent*1.01, vmax=max_sent*1.01)

sentiment_colors = [cmap(norm(i)) for i in sentiments]

color = []
i = 0
for node in Gg.nodes:
  if node == ego_name:
    color.append(cmap2.colors[i])
  else:
    color.append(cmap2.colors[i+1])

sm = plt.cm.ScalarMappable(cmap=cmap, norm=plt.Normalize(vmin=min_sent, vmax=max_sent))
sm.set_array([])
cbar = plt.colorbar(sm)
cbar.ax.set_ylabel('Sentiment',size=30,rotation=270,labelpad=20)

node_size = []
for node in Gg.nodes:
  if node != ego_name:
    node_size.append(n_dic[node]['size']*node_scalar)
  else:
    node_size.append(n_dic[node]['size']*node_scalar*1.2)

# node_and_degree = Gg.degree()
# (largest_hub, degree) = sorted(node_and_degree, key=itemgetter(1))[-1]
largest_hub=ego_name
# Create ego graph of main hub
hub_ego = nx.ego_graph(Gg, largest_hub)
# Draw graphu
pos = nx.spring_layout(hub_ego,3)
#hub_ego.remove_node(ego_name)
nx.draw(hub_ego, pos, node_color=cmap2.colors[4], node_size=node_size, with_labels=True,font_size=12,font_weight='bold',width=[math.log
(e_dic[i]['width']+1)*edge_scalar for i in Gg.edges],edge_color=sentiment_colors)
# Draw ego as large and red
```

```
nx.draw_networkx_nodes(hub_ego.remove_node(ego_name), pos, nodelist=[largest_hub], node_size=n_dic[ego_name]['size']*node_scalar, n
ode_color=cmap2.colors[3], with_labels=True,font_size=12,font_weight='bold')

plt.title('Egocentric Sentential Co-Occurrence of {}'.format(ego_name),size=40)

plt.subplots(figsize=(15,15))

node_scalar = 250
edge_scalar = 3

ego_name = 'Annex'

Gg,n_dic,e_dic = make_G(df,'sentences',6,node_dic,sent_dic,sentiments2,ego_name,'')

#pos = graphviz_layout(G)

sentiments = [e_dic[i]['sentiment'] for i in Gg.edges]

if min(sentiments) >= 0:
  min_sent = -1
else:
  min_sent = min(sentiments)

if max(sentiments) <= 0:
  max_sent = 1
else:
  max_sent = max(sentiments)

cmap = cm.get_cmap('viridis_r')
cmap2 = cm.get_cmap('Set3')

norm = matplotlib.colors.DivergingNorm(vcenter=0, vmin=min_sent*1.01, vmax=max_sent*1.01)

sentiment_colors = [cmap(norm(i)) for i in sentiments]

color = []
i = 0
for node in Gg.nodes:
  if node == ego_name:
    color.append(cmap2.colors[i])
  else:
    color.append(cmap2.colors[i+1])

sm = plt.cm.ScalarMappable(cmap=cmap, norm=plt.Normalize(vmin=min_sent, vmax=max_sent))
sm.set_array([])
cbar = plt.colorbar(sm)
cbar.ax.set_ylabel('Sentiment',size=30,rotation=270,labelpad=20)

node_size = []
for node in Gg.nodes:
  if node != ego_name:
    node_size.append(n_dic[node]['size']*node_scalar)
  else:
    node_size.append(n_dic[node]['size']*node_scalar*1.2)

# node_and_degree = Gg.degree()
# (largest_hub, degree) = sorted(node_and_degree, key=itemgetter(1))[-1]
largest_hub = ego_name
# Create ego graph of main hub
hub_ego = nx.ego_graph(Gg, largest_hub)
# Draw graphu
pos = nx.spring_layout(hub_ego,3)
#hub_ego.remove_node(ego_name)
nx.draw(hub_ego, pos, node_color=cmap2.colors[4], node_size=node_size, with_labels=True,font_size=12,font_weight='bold',width=[math.log
(e_dic[i]['width']+1)*edge_scalar for i in Gg.edges],edge_color=sentiment_colors)
# Draw ego as large and red
```

```
nx.draw_networkx_nodes(hub_ego.remove_node(ego_name), pos, nodelist=[largest_hub], node_size=n_dic[ego_name]['size']*node_scalar, n
ode_color=cmap2.colors[3], with_labels=True,font_size=12,font_weight='bold')

plt.title('Egocentric Sentential Co-Occurrence of {}'.format(ego_name),size=40)
```

8. Graph Partitions
8a. Partition of Fluid Communities

```
commsd_fl = {}
for z,v in color_comm_membership.items():
  commsd_fl[z] = len(v)
ddf=pd.DataFrame(commsd_fl.items(), columns=['Community', 'Number of Nodes'])
sst="Bar plot of Fluid Communities"
ddf.plot.bar(x='Community', y='Number of Nodes', figsize=(12,7),rot=0,title=sst);
#G,n_dic,e_dic = make_G(df,'sentences',14,node_dic,sent_dic,sentiments2,'','Anne')

node_color = []
for node in G.nodes:
  for z,v in color_comm_membership.items():
    if node in v:
      node_color.append(z)

vmin = min(node_color) #df['color'].min()
vmax = max(node_color) #df['color'].max()
cmap=plt.cm.viridis #plt.cm.coolwarm #plt.cm.Blues #

colors = [i for i in node_color]

plt.figure(figsize=(10,7));
node_border_color='k'
nodes = nx.draw_networkx_nodes(G, final_pos_dic, node_color=colors, cmap=cmap, vmin=vmin, vmax=vmax)
nodes.set_edgecolor(node_border_color)
nx.draw_networkx_edges(G, final_pos_dic,edge_color='b',alpha=0.8)
plt.axis('off');
yoffset = {}
y_off = -0.7 #0.05  # offset on the y axis
for z, v in final_pos_dic.items():
    yoffset[z] = (v[0], v[1]+y_off)
nx.draw_networkx_labels(G, yoffset,font_weight='bold',font_size='11');
sm = plt.cm.ScalarMappable(cmap=cmap, norm=plt.Normalize(vmin=vmin, vmax=vmax))
sm.set_array([])
cbar = plt.colorbar(sm,label="Fluid Communities") #, orientation='horizontal')
sst="Fluid Communities Clustered by NLP Attribute"
plt.title(sst,fontsize=15);
plt.margins(x=0.1, y=0.1)
```

8b. Partition of Lovuain Communities

```
ddf=pd.DataFrame(commsd_lo.items(), columns=['Community', 'Number of Nodes'])
sst="Bar plot of Louvain Communities"
ddf.plot.bar(x='Community', y='Number of Nodes', figsize=(12,7),rot=0,title=sst);
#G,n_dic,e_dic = make_G(df,'sentences',14,node_dic,sent_dic,sentiments2,'','Anne')

partition_lo = community.best_partition(G)
node_color=partition_lo.values()
vmin = min(node_color) #df['color'].min()
vmax = max(node_color) #df['color'].max()
cmap=plt.cm.viridis #plt.cm.coolwarm #plt.cm.Blues #

colors = [i for i in node_color]

plt.figure(figsize=(10,7));
node_border_color='k'
nodes = nx.draw_networkx_nodes(G, final_pos_dic, node_color=colors, cmap=cmap, vmin=vmin, vmax=vmax)
nodes.set_edgecolor(node_border_color)
nx.draw_networkx_edges(G, final_pos_dic,edge_color='b',alpha=0.8)
```

```
plt.axis('off');
yoffset = {}
y_off = -0.7 #0.05  # offset on the y axis
for z, v in final_pos_dic.items():
   yoffset[z] = (v[0], v[1]+y_off)
nx.draw_networkx_labels(G, yoffset,font_weight='bold',font_size='11');
sm = plt.cm.ScalarMappable(cmap=cmap, norm=plt.Normalize(vmin=vmin, vmax=vmax))
sm.set_array([])
cbar = plt.colorbar(sm,label="Louvain Communities") #, orientation='horizontal')
sst="Louvain Communities Clustered by NLP Attribute"
plt.title(sst,fontsize=15);
plt.margins(x=0.1, y=0.1)
```

8c. Partition of Leiden Communities

```
ddf=pd.DataFrame(commsd_le.items(), columns=['Community', 'Number of Nodes'])
sst="Bar plot of Leiden Communities"
ddf.plot.bar(x='Community', y='Number of Nodes', figsize=(12,7),rot=0,title=sst);

#G,n_dic,e_dic = make_G(df,'sentences',14,node_dic,sent_dic,sentiments2,'','Anne')

node_color = []
for node in G.nodes:
  for z,v in leiden_comm_membership.items():
    if node in v:
      node_color.append(z)

vmin = min(node_color) #df['color'].min()
vmax = max(node_color) #df['color'].max()
cmap=plt.cm.viridis #plt.cm.coolwarm #plt.cm.Blues #

for part_list in partition_le:
  part = [list(G.nodes())[i] for i in part_list]
  partition_nodes.append(part)

colors = [i for i in node_color]

plt.figure(figsize=(10,7));
node_border_color='k'
nodes = nx.draw_networkx_nodes(G, final_pos_dic, node_color=colors, cmap=cmap, vmin=vmin, vmax=vmax)
nodes.set_edgecolor(node_border_color)
nx.draw_networkx_edges(G, final_pos_dic,edge_color='b',alpha=0.8)
plt.axis('off');
yoffset = {}
y_off = -0.7 #0.05  # offset on the y axis
for z, v in final_pos_dic.items():
   yoffset[z] = (v[0], v[1]+y_off)
nx.draw_networkx_labels(G, yoffset,font_weight='bold',font_size='11');
sm = plt.cm.ScalarMappable(cmap=cmap, norm=plt.Normalize(vmin=vmin, vmax=vmax))
sm.set_array([])
cbar = plt.colorbar(sm,label="Leiden Communities") #, orientation='horizontal')
sst="Leiden Communities Clustered by NLP Attribute"
plt.title(sst,fontsize=15);
plt.margins(x=0.1, y=0.1)
```

8d. Community Symmetric Differences

```
def ccomp(A,type1,B,type2):
   eds=[]
   A1=set(A.keys())
   B1=set(B.keys())
   for i in A1:
      x=set(A[i])
      for j in B1:
         y=set(B[j])
         sd=sorted(list(x.symmetric_difference(y)))
```

```
            ss=sorted(list(set(list(x)+list(y))))
            if sorted(A[i])==sorted(B[j]):
                eds.append((type1+str(i),type2+str(j),1.))
            else:
                if len(sd)/float(len(ss))==1:
                    eds.append((type1+str(i),type2+str(j),0.))
                else:
                    eds.append((type1+str(i),type2+str(j),len(sd)/float(len(ss))))
    H = nx.Graph()
    H.add_weighted_edges_from(eds)
    H.remove_edges_from([e for e in H.edges(data=True) if e[2]['weight']==0.])
    return H

fluid_partition = {}
for k,v in color_comm_membership.items():
  for x in v:
    fluid_partition[x] = k

type1='Louvain_Comm_'
type2='Leiden_Comm_'
type3='Fluid_Comm_'
H=ccomp(louvain_comm_membership,type1,leiden_comm_membership,type2)
H1=ccomp(louvain_comm_membership,type1,color_comm_membership,type3)
H2=ccomp(leiden_comm_membership,type2,color_comm_membership,type3)
H.add_nodes_from(H1.nodes)
H.add_nodes_from(H2.nodes)
H.add_weighted_edges_from([(i[0],i[1],H1.edges[i]['weight']) for i in H1.edges])
H.add_weighted_edges_from([(i[0],i[1],H2.edges[i]['weight']) for i in H2.edges])

edge_scalar = 10

posbp={}
louvain=sorted([n for n in H.nodes() if "Louvain" in n])
leiden=sorted([n for n in H.nodes() if "Leiden" in n])
fluid=sorted([n for n in H.nodes() if "Fluid" in n])

i=0
for node in louvain:
  xs = np.linspace(1.0, 5.0, num=len(louvain))
  posbp[node] = (xs[i],0)
  i += 1

i=0
for node in leiden:
  xs = np.linspace(0.5, 2, num=len(leiden))
  posbp[node] = (xs[i],-0.57*xs[i])
  i += 1

i=0
for node in fluid:
  xs = np.linspace(4, 6.0, num=len(fluid))
  posbp[node] = (xs[i],0.57*xs[i]-3.7)
  i += 1

elabels={}
elabels = nx.get_edge_attributes(H,'weight')
elabelsc={}
for z,v in elabels.items():
    elabelsc[z]='%.02f' %v
elabels=elabelsc

plt.figure(figsize=(12,10));
widths = [float(i) for i in list(elabels.values())]
widths = [i/max(widths)*edge_scalar for i in widths]
nx.draw(H, posbp, edgelist=H.edges(), edge_color=widths, width=10.0, edge_cmap=plt.cm.Blues)
nodes1 = nx.draw_networkx_nodes(H, posbp, nodelist=louvain,node_color="b",node_shape='s',alpha=0.4)
```

```
nodes2 = nx.draw_networkx_nodes(H, posbp, nodelist=leiden,node_color="g",node_shape='s',alpha=0.4)
nodes3 = nx.draw_networkx_nodes(H, posbp, nodelist=fluid,node_color="r",node_shape='s',alpha=0.4)
#nx.draw_networkx_edges(H, posbp, edge_colors=widths,width=4, alpha=0.7,edge_cmap=plt.cm.Blues)
# nx.draw_networkx_labels(H, posbp)
#nx.draw_networkx_edge_labels(H,posbp,edge_labels=elabels,label_pos = 0.7,font_size=7);
plt.axis('off');
yoffset = {}
y_off = -0.05  # offset on the y axis
for z, v in posbp.items():
   yoffset[z] = (v[0], v[1]+y_off)
nx.draw_networkx_labels(H, yoffset,font_weight='bold');
plt.margins(x=0.1, y=0.1)
plt.title("Similarities-Weighted Communities\nfor the Multilayer Graph (1944)",size=40);
```

8f. Modularity Comparison by Community

```
modularity_fl=community.modularity(fluid_partition, G, weight='weight')
import networkx.algorithms.community.quality as qu
loucomms=[v for k,v in louvain_comm_membership.items()]
leicomms=[v for k,v in leiden_comm_membership.items()]
fluidcomms=[v for k,v in color_comm_membership.items()]
perf_lo=qu.performance(G,loucomms)
perf_le=qu.performance(G,leicomms)
perf_fl=qu.performance(G,fluidcomms)
cov_lo=qu.coverage(G,loucomms)
cov_le=qu.coverage(G,leicomms)
cov_fl=qu.coverage(G,fluidcomms)
plt.style.use('seaborn-dark')

labels = ['Louvain','Leiden','Fluid']
modularities = [modularity_lo,modularity_le,modularity_fl]
performances = [perf_lo,perf_le,perf_fl]
coverage = [cov_lo,cov_le,cov_fl]

x = np.arange(len(labels))  # the label locations
width = 0.25  # the width of the bars
buf = 0.1

fig, ax = plt.subplots()
rects1 = ax.bar(x - width, modularities, width, label='Modularity')
rects2 = ax.bar(x, performances, width, label='Performance')
rects3 = ax.bar(x + width, coverage, width, label='Coverage')

# Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_ylabel('Metric Score',size=18)
ax.set_title('Community Metrics',size=24)
ax.set_xticks(x)
ax.set_xticklabels(labels,size=18)
ax.legend()

fig.tight_layout()

fig.set_figheight(10)
fig.set_figwidth(8)
plt.axhline(y=0, color='k', linestyle='-')
plt.show()
```

9. Cliques

9a. Clique Identification

```
G_cl,n_dic_cl,e_dic_cl = make_G(df,'sentences',10,node_dic,sent_dic,sentiments2,'','')

cliques=[clique for clique in nx.find_cliques(G_cl)]
gcliques=[clique for clique in cliques if len(clique)>2]
lcliques=[clique for clique in cliques if len(clique)<=2]
print("The total number of cliques is %i" %len(cliques))
```

```
print("The total number of non trivial cliques is %i" %len(gcliques))
print("The total number of trivial cliques is %i" %len(lcliques))
```

9b. Bipartite Clique Graph

```
n2cdi={}
for n in G_cl.nodes():
    for w in gcliques:
        if n in w:
            n2cdi[n]=w

cdi={}
for i,j in list(enumerate(cliques)):
    if i+1<10:
        cdi["clique_0"+str(i+1)]=j
    else:
        cdi["clique_"+str(i+1)]=j

gcdi={k:v for k,v in cdi.items() if len(v)>2}

eds=[]
for k,v in cdi.items(): #g
    for vv in v:
        eds.append((k,vv))
H=nx.Graph()
H.add_edges_from(eds)
Y=sorted([n for n in list(H.nodes()) if type(n)==str and "clique_" in n])
X=sorted([n for n in H.nodes() if n not in Y])

pu=1.

posbp={}
if len(X)==max([len(X),len(Y)]):
    for i,x in list(enumerate(X)):
        posbp[x]=(0,i)
    for j,y in list(enumerate(Y)):
        posbp[y]=(1,(j+j*float(i))/len(Y))
else:
    for i,x in list(enumerate(Y)):
        posbp[x]=(1,i)
    for j,y in list(enumerate(X)):
        posbp[y]=(0,(j+j*float(i))/len(Y))

plt.figure(figsize=(7,10));
nodes1 = nx.draw_networkx_nodes(H, pos=posbp,node_size=100,nodelist=Y,node_color="#b3ffb3",node_shape='s')
nodes2 = nx.draw_networkx_nodes(H, pos=posbp,node_size=100,nodelist=X,node_color="#ffb3b3",node_shape='o')

nx.draw_networkx_edges(H, pos=posbp, edge_color="b", alpha=0.5)
nx.draw_networkx_labels(H, pos=posbp,font_size=10)
plt.axis('off');

plt.margins(x=0.25, y=0.1)
sst="The bipartite graph of nodes vs. cliques"
plt.title(sst);
```

9c. Clique Intersection

```
from itertools import combinations

cleds=[]
# sbeds=[]
for k,v in gcdi.items():
    ve=G_cl.subgraph(v).edges()
    for w in ve:
        cleds.append(w)
cleds+=[(c2,c1) for (c1,c2) in cleds]
```

```
sbeds=[e for e in G_cl.edges() if e not in cleds]

ceds=[]
for c in list(combinations(Y, 2)):
   ints=set(cdi[c[0]]).intersection(cdi[c[1]])
   if len(ints)>0:
      ceds.append((c[0],c[1],len(ints),list(ints)))

weight={(i,j):(k,l) for (i,j,k,l) in ceds}

w_edges=[(x,y,z) for (x,y),z in weight.items()]

cG = nx.Graph()
cG.add_weighted_edges_from(w_edges)

elabels={}
elabels = nx.get_edge_attributes(cG,'weight')
edge_width=[cG[u][v]['weight'][0] for u,v in cG.edges()]
edge_width=[w for w in edge_width]

lcdi={k:len(v) for k,v in cdi.items()}
nx.set_node_attributes(cG, lcdi, name="clique_size")

nodes_between_2cliques={}
tt=[]
for (x,y,z) in cG.edges(data=True):
   for xx in cdi[x]:
      if xx not in n2cdi.keys():
         for yy in cdi[y]:
            if yy not in n2cdi.keys():
               tt.append((xx,yy,z))
for (x,y,z) in tt:
   if z['weight'][0]==1:
      if z['weight'][1][0] not in n2cdi.keys():
#        if lcdi[x]==2 and lcdi[y]==2:
         nodes_between_2cliques[(x,y)]=z['weight'][1][0]
# print len(set(nodes_between_2cliques.values())),list(set(nodes_between_2cliques.values()))

figsize=(15,15)
pos=graphviz_layout(cG); #pos=nx.circular_layout(G);
node_color="#b3ffb3" #"#ffb3b3"
node_border_color="g"
edge_color="#668cff"
plt.figure(figsize=figsize);
nodes = nx.draw_networkx_nodes(cG, pos, node_size=[100*v for v in lcdi.values()],node_color=node_color,node_shape="s")
nodes.set_edgecolor(node_border_color)
nx.draw_networkx_edges(cG, pos, edge_color=edge_color,width=edge_width)
nx.draw_networkx_labels(cG, pos)
# nx.draw_networkx_edge_labels(cG,pos,edge_labels=elabels);
plt.axis('off');
sst="The intersection graph of cliques"
plt.title(sst);
```

9d. Clustering of Clique Intersection

```
intersecting_clique_edges=[]
for e in cG.edges(data=True):
   a=cdi[e[0]]
   b=cdi[e[1]]
   c=set(a).intersection(set(b))
   if len(c)>1:
      sG=G_cl.subgraph(list(c))
      for se in sG.edges():
         if se not in intersecting_clique_edges:
            intersecting_clique_edges.append(se)
```

```
leaves=[n for n in G_cl.nodes() if G_cl.degree(n)==1]

pendants=[e for e in G_cl.edges() if e[0] in leaves or e[1] in leaves]

nb2c=list(set(nodes_between_2cliques.values()))

pos=graphviz_layout(G_cl,'neato')
plt.figure(figsize=(15,15));

nx.draw_networkx_edges(G_cl, pos, edgelist=[e for e in sbeds if e not in pendants],width=3,edge_color="r", alpha=0.7)
nx.draw_networkx_edges(G_cl, pos, edgelist=pendants,width=3,edge_color="orange", alpha=0.5)
nx.draw_networkx_edges(G_cl, pos, edgelist=[e for e in G_cl.edges() if e not in sbeds and e not in pendants and e not in intersecting_clique_ed
ges],width=3,edge_color="c", alpha=0.7)
nx.draw_networkx_edges(G_cl, pos, edgelist=intersecting_clique_edges,width=3,edge_color="b", alpha=0.8)
nodes = nx.draw_networkx_nodes(G_cl, pos, nodelist=[n for n in G_cl.nodes() if n not in nb2c and n not in leaves],node_size=500,node_color="
r", alpha=0.9)
nodes = nx.draw_networkx_nodes(G_cl, pos, nodelist=nb2c,node_size=500,node_color="g", alpha=0.9)
nodes = nx.draw_networkx_nodes(G_cl, pos, nodelist=leaves,node_size=500,node_color="orange", alpha=0.9)

plt.axis('off');
yoffset = {}
y_off = - 15
for k, v in pos.items():
   yoffset[k] = (v[0], v[1]+y_off)
nx.draw_networkx_labels(G_cl, yoffset,font_weight='bold',font_size=13);
plt.margins(x=0.25, y=0.2)
sst="Cliques of Sentential Co-
Occurrence (1944)"#\n green nodes = brokers between cliques \n orange nodes = leaves \n red edges = bridges between cliques \n cyan edges
= non-intersecting clique edges \n blue edges = intersecting clique edges"
plt.title(sst,size=35);
```

10. Network Statistics
10a. Centralities

```
def create_centralities_list(G_cl,maxiter=2000,pphi=5,centList=[]):
   if len(centList)==0:
      centList=['degree','closeness','betweenness','eigenvector','Katz','PageRank','HITS','load','communicability','current flow']
   cenLen=len(centList)
   valus={}
   for uu,centr in enumerate(centList):
      if centr=='degree':
         if isinstance(G_cl,nx.DiGraph):
            cent=nx.in_degree_centrality(G_cl)
            sstt='In Degree Centralities '
            valus['in_degree']=cent
            cent=nx.out_degree_centrality(G_cl)
            sstt+= 'and Out Degree Centralities'
            valus['out_degree']=cent
         else:
            cent=nx.degree_centrality(G_cl)
            sstt='Degree Centralities'
            ssttt='degree centrality'
            valus[centr]=cent
      elif centr=='closeness':
         cent=nx.closeness_centrality(G_cl)
         sstt='Closeness Centralities'
         ssttt='closeness centrality'
         valus[centr]=cent
      elif centr =='load':
         cent=nx.load_centrality(G_cl)
         sstt='Load Centraities'
         valus[centr]=cent
      elif centr == 'communicability':
         if not isinstance(G_cl, nx.DiGraph):
```

```
            cent=nx.communicability_betweenness_centrality(G_cl)
            sstt='Communicability Centralities'
            valus[centr]=cent
        elif centr=='betweenness':
            cent=nx.betweenness_centrality(G_cl)
            sstt='Betweenness Centralities'
            ssttt='betweenness centrality'
            valus[centr]=cent
        elif centr=='current flow':
            if not isinstance(G_cl, nx.DiGraph):

                cent=nx.current_flow_closeness_centrality(G_cl)
                sstt='Current Flow Closeness Centrality'
                valus[centr]=cent
        elif centr=='eigenvector':
            try:
                cent=nx.eigenvector_centrality(G_cl,max_iter=maxiter)
                sstt='Eigenvector Centralities'
                ssttt='eigenvector centrality'
                valus[centr]=cent

            except:
                valus[centr]=None

                continue
        elif centr=='Katz':
            phi = (1+math.sqrt(pphi))/2.0 # largest eigenvalue of adj matrix
            cent=nx.katz_centrality_numpy(G_cl,1/phi-0.01)
            cent=nx.katz_centrality_numpy(G_cl,.05)#,1/phi-0.01)

            sstt='Katz Centralities'
            ssttt='Katz centrality'
            valus[centr]=cent
#           valus[centr+'_%i' %pphi]=cent

        elif centr=='PageRank':
            try:
                cent=nx.pagerank(G_cl)
                sstt='PageRank'
                ssttt='pagerank'
                valus[centr]=cent

            except:
                valus[centr]=None

                continue
        elif centr=='HITS':
            if isinstance(G_cl,nx.DiGraph):
                dd=nx.hits(G_cl,max_iter=maxiter)
                sstt='HITS hubs '
                valus['HITS_hubs']=dd[0]
                sstt+= 'and HITS authorities'
                valus['HITS_auths']=dd[1]
            else:
                dd=nx.hits(G_cl,max_iter=maxiter)
                cent=nx.degree_centrality(G_cl)
                sstt='HITS'
                ssttt='HITS Centralities'
                valus[centr]=dd[0]
        else:
            continue
        print('%s done!!!' %sstt)
    return valus

dindices=['out_degree','in_degree','closeness','betweenness','eigenvector','HITS_hubs','HITS_auths','Katz','PageRank','load']
indices=['degree','closeness','betweenness','eigenvector','HITS','Katz','PageRank','load','communicability','current flow']
```

```
# indices=['degree','closeness','betweenness','eigenvector']

# Without 'communicability' and 'current flow' (undirected case)
dindicesd=['out_degree','in_degree','closeness','betweenness','eigenvector','HITS_hubs','HITS_auths','Katz','PageRank','load']
indicesd=['degree','closeness','betweenness','eigenvector','HITS','Katz','PageRank','load']
# indicesd=['degree','closeness','betweenness','eigenvector']

dindicesdr=dindices
indicesdr=indices

# Plus 'node'
dindicesdrn=["node"]+dindices
indicesdrn=['node']+indices

def picker(G_cl,data,cols,thres):
    for al in range(1,len(G_cl.nodes())+1):
        degs_dic={}
        deggs=None
        ndegs_dic={}
        for col in cols:
            if col not in ndegs_dic:
                ndegs_dic[col]={}
            degress=central_pd.sort_values(col,ascending=False).head(al).to_dict()
            if deggs is None:
                deggs=set(degress[col].keys())
            else:
                deggs=deggs.intersection(set(degress[col].keys()))
            degs_dic[col]=degress[col]#.keys()
            for ije, vvf in enumerate(sorted(set(degress[col].values()),reverse=True)):
                ndegs_dic[col][vvf]=ije+1
        if len(deggs)>=thres:
            break

    rank_dic=Counter()
    rrank_dic={}
    for i in deggs:
        ss='Key-player node %s is ranked: \n' %i  #has ranking index
        if i not in rrank_dic:
            rrank_dic[i]=[]
        for nn,kk in degs_dic.items():
            ss+='%i-th wrt %s (%f)\n' %(ndegs_dic[nn][kk[i]],nn,kk[i])#(kk.index(i)+1,nn)
            rank_dic[i]+=ndegs_dic[nn][kk[i]]
            rrank_dic[i].append(ndegs_dic[nn][kk[i]])
#       print ss
    return al,deggs,degs_dic,rank_dic,rrank_dic,ndegs_dic

central_pd=pd.DataFrame(create_centralities_list(G_cl))
central_pd["node"]=central_pd.index.values
if isinstance(G_cl,nx.DiGraph):
    central_pd=central_pd[['node']+dindices]
else:
    central_pd=central_pd[['node']+indices]
central_pd["node"]=central_pd.index.values
central_pd.reset_index(drop = True, inplace = True)
central_pd.sort_values('node') #.head()
central_pd

data = central_pd

if isinstance(G_cl,nx.DiGraph):
#     cols=dindicesdr# no hits
    cols=dindices
else:
#     cols=indicesdr#no hits
    cols=indices
```

```
thres=len(G_cl.nodes())

# al,deggs,degs_dic,rankings,rankd = picker(G,data,cols,thres) #,thres=1)
al,deggs,degs_dic,rankings,rrank_dic,ndegs_dic = picker(G_cl,data,cols,thres) #,thres=1)

central_pd=data.set_index('node')
nndegs_dic={}
ppdff=central_pd.to_dict(orient='dict')

for k,v in ppdff.items():
   nndegs_dic[k]=[ndegs_dic[k][j] for j in v.values()]

nndegs_dic['node']=[j.keys() for j in ppdff.values()][0]

fpdngegs=pd.DataFrame.from_dict(nndegs_dic, orient='index')
fpdngegs=fpdngegs.T
fpdngegs

if isinstance(G_cl,nx.DiGraph):
   df_c=fpdngegs[dindices]
else:
   df_c=fpdngegs[indices]

rd={}
for i in df_c.index.values:
   vv=df_c.iloc[i].tolist()
   ra=min(vv)
   vv=len(df_c.columns)*[ra]
   if isinstance(G_cl,nx.DiGraph):
      dff=df_c[(df_c['out_degree'] >= vv[0]) & (df_c['in_degree'] >= vv[1]) & (df_c['closeness'] >= vv[2]) & (df_c['betweenness'] >= vv[3]) & (df_c['
eigenvector'] >= vv[4]) & (df_c['HITS_hubs'] >= vv[5]) & (df_c['HITS_auths'] >= vv[6]) & (df_c['Katz'] >= vv[7]) & (df_c['PageRank'] >= vv[8]) & (df
_c['load'] >= vv[9])]
   else:
      dff=df_c[(df_c['degree'] >= vv[0]) & (df_c['closeness'] >= vv[1]) & (df_c['betweenness'] >= vv[2]) & (df_c['eigenvector'] >= vv[3]) & (df_c['Kat
z'] >= vv[4]) & (df_c['PageRank'] >= vv[5]) & (df_c['load'] >= vv[6]) & (df_c['communicability'] >= vv[7]) & (df_c['current flow'] >= vv[8])]
   rd[i]=len(dff)

mrd={}
for i,v in rd.items():
   mrd[fpdngegs.iloc[i]["node"]]=max(rd.values())-rd[i]+1
mrd

nn=[]
rn=[]
ru=[]
rankings=rd
en=enumerate(sorted(rankings, key=rankings.get, reverse=False))
r = {key: rank for rank, key in enumerate(sorted(set(rankings.values()), reverse=False), 1)}
rdd={k: r[v] for k,v in rankings.items()}
for i, w in en:
   nn.append(fpdngegs.iloc[w]['node'])
   rn.append(max(rdd.values())-rdd[w]+1)
   ru.append(rankings[w])
rdf=pd.DataFrame(
   {'node': nn,
    'rung' : ru,
    'rank': rn
   })
rdf=rdf[["node","rung","rank"]]
rdf.sort_values("rank", inplace=True)
rdf

rdf[indices]=fpdngegs[indices]
import seaborn as sns
mrdf = rdf
cm = sns.light_palette("green", as_cmap=True)
```

```
mrdf[['degree','closeness','betweenness','eigenvector','HITS','Katz','PageRank','load','communicability','current flow']] = mrdf[['degree','closene
ss','betweenness','eigenvector','HITS','Katz','PageRank','load','communicability','current flow']].apply(pd.to_numeric)
mrdf.style.background_gradient(cmap='viridis')

clr={}
for i in range(len(rdf)):
   clr[rdf.iloc[i]["node"]]=rdf.iloc[i]["rank"]
node_color=[clr[n] for n in G_cl.nodes()]

nsi=[50*G_cl.degree(n) for n in G_cl.nodes()]

labels={}
for i in G_cl.nodes():
   labels[i]=i

edge_scalar = 2
node_scalar = 3
nsi = [i*node_scalar for i in nsi]
edge_width = [math.log(e_dic_cl[i]['width']+1)*edge_scalar for i in G_cl.edges]

plt.figure(figsize=(18,10))
nx.draw_networkx_edges(G_cl,pos=pos,width=edge_width, edge_color="lime", alpha=0.8);
nc=nx.draw_networkx_nodes(G_cl,pos=pos,cmap=plt.get_cmap('rainbow'), node_color=node_color,edgecolors='r',node_size=nsi,alpha=1);
nx.draw_networkx_labels(G_cl,pos=pos,labels=labels,font_size=12);

cbar=plt.colorbar(nc)
#cbar.set_label('Aggregated Centralities Rating')
cbar.ax.set_ylabel('Aggregated Centralities Rating',size=20,rotation=270,labelpad=30)
plt.axis("off");
tt="Sentential Co-Occurrence\nNodes Colored According to Aggregated Centrality Rating"
plt.title(tt,size=25);

# name = 'Sentential Co-Occurrences'

# df4 = df.loc[list(G_cl.nodes),list(G_cl.nodes)].copy()
# axes = pd.plotting.scatter_matrix(df4, figsize=(15, 15), marker='o',hist_kwds={'bins': 20}, s=60, alpha=.8)
# corr = df4.corr().values
# for i, j in zip(*plt.np.triu_indices_from(axes, k=1)):
#    axes[i, j].annotate("%.3f" %corr[i,j], (0.8, 0.8), xycoords='axes fraction', ha='left', va='center',rotation='vertical')
# sst="%s \n Scatter Matrix of Centrality Indices \n (displaying Pearson correlation coefficients)" %name

# n = len(G_cl.nodes)

# for x in range(n):
#    for y in range(n):
#       # to get the axis of subplots
#       ax = axes[x, y]
#       # to make x axis name vertical
#       ax.xaxis.label.set_rotation(90)
#       # to make y axis name horizontal
#       ax.yaxis.label.set_rotation(0)
#       # to make sure y axis names are outside the plot area
#       ax.yaxis.labelpad = 20

# plt.suptitle(sst,fontsize=20);

name = 'Sentential Co-Occurrence'
pos=graphviz_layout(G_cl,'circo')
valus=create_centralities_list(G_cl,maxiter=2000,pphi=5,centList=[])
if isinstance(G_cl,nx.DiGraph):
   cts=['out_degree','in_degree','closeness','betweenness','eigenvector','HITS_hubs','HITS_auths','Katz','PageRank','load']
else:
   cts=['degree','closeness','betweenness','eigenvector','HITS','Katz','PageRank','load','communicability','current flow']
for cent in valus.values():
   cs={}
   for k,v in cent.items():
```

```
        if v not in cs:
            cs[v]=[k]
        else:
            cs[v].append(k)
    nodrank=[]
    uui=0
    for k in sorted(cs,reverse=True):
        for v in cs[k]:
            if uui<len(G_cl): #5:
                nodrank.append(v)
                uui+=1
    nodeclo=[]
    for k,v in cent.items():
        if k in  nodrank :
            nodeclo.append(v)
        else:
            nodeclo.append(0.)
plt.figure(figsize=(25,50))
for i,j in enumerate(cts):
    plt.subplot(len(cts)/2,2,i+1).set_title(j)
    nx.draw_networkx_nodes(G_cl,pos=pos,nodelist=valus[j].keys(),
                           node_size=[500*x for x in valus[j].values()],
                           node_color=nodeclo,
                           cmap=plt.cm.cool,alpha=0.7) #Reds
    nx.draw_networkx_edges(G_cl,pos=pos,edge_color='b', alpha=0.5)
    y_off = 50 #0.05  # offset on the y axis
    pos_off = {key: (pos[key][0],pos[key][1] - y_off) for key in pos}
    nx.draw_networkx_labels(G_cl,pos=pos_off,font_weight='bold',font_size='10');
    nx.draw_networkx_edges(G_cl,pos=pos,edge_color='b', alpha=0.5)
    plt.title(j,fontsize=20)
    kk=plt.axis('off')
sst="%s Centrality Indices" %name
plt.suptitle(sst,fontsize=20);

comm_dic = nx.communicability(G_cl)
comm_dic2 = {}
for k,z in comm_dic.items():
  comm_dic2[k] = {}
  comm_dic2[k]['communicability'] = {}
  data_list = []
  for k1,z1 in z.items():
    data_list.append(z1)
  comm_dic2[k]['communicability'] = round(sum(data_list)/len(data_list),3)
  sent_avg = []
  widths2 = []
  for e in e_dic_cl.keys():
    if (k in e[0]) or (k in e[1]):
      sent_avg.append(e_dic_cl[e]['sentiment']*e_dic_cl[e]['width'])
      widths2.append(e_dic_cl[e]['width'])
  comm_dic2[k]['avg_sentiment'] = round(sum(sent_avg)/(sum(widths2)*len(sent_avg)),3)
comm_dic2

yd = [comm_dic2[k]['avg_sentiment'] for k,z in comm_dic2.items()]
xd = [comm_dic2[k]['communicability'] for k,z in comm_dic2.items()]

plt.figure(figsize=(10,10))

# make the scatter plot
plt.scatter(xd, yd, s=30, alpha=0.75, marker='o')

# determine best fit line
par = np.polyfit(xd, yd, 1, full=True)

slope=par[0][0]
intercept=par[0][1]
xl = [min(xd), max(xd)]
```

```
yl = [slope*xx + intercept  for xx in xl]

# coefficient of determination, plot text
variance = np.var(yd)
residuals = np.var([(slope*xx + intercept - yy)  for xx,yy in zip(xd,yd)])
Rsqr = np.round(1-residuals/variance, decimals=2)
plt.text(.4*max(xd)+.1*min(xd),0.2*max(yd)+.1*min(yd),'$R^2 = %0.2f$'% Rsqr, fontsize=30)

plt.ylabel("Weighted Sentiment",size=20)
plt.xlabel("Communicability",size=20)
plt.title('Weighted Sentiment vs. Communicability',size=25)

# error bounds
yerr = [abs(slope*xx + intercept - yy)  for xx,yy in zip(xd,yd)]
par = np.polyfit(xd, yerr, 2, full=True)

# yerrUpper = [(xx*slope+intercept)+(par[0][0]*xx**2 + par[0][1]*xx + par[0][2]) for xx,yy in zip(xd,yd)]
# yerrLower = [(xx*slope+intercept)-(par[0][0]*xx**2 + par[0][1]*xx + par[0][2]) for xx,yy in zip(xd,yd)]

for k,z in comm_dic.items():
  y,x = comm_dic2[k]['avg_sentiment'],comm_dic2[k]['communicability']
  if (abs(y) > 0.08) or (k=='Dutch'):
    plt.text(x+.01,y+.02, k, fontsize=12,fontweight='bold')

plt.plot(xl, yl, '-r')
plt.axhline(y=0, color='k', linestyle='-')
# plt.plot(xd, yerrLower, '--r')
# plt.plot(xd, yerrUpper, '--r')
plt.show()

xd_list = [i/(len(G.nodes)*len(G.edges)) for i in xd]
xd_list = [(i-min(xd_list)) for i in xd_list]
#xd_list = xd
yd_list = yd
xd_list

clos_dic = nx.closeness_centrality(G_cl)
yd = [comm_dic2[k]['avg_sentiment'] for k,z in clos_dic.items()]
xd = [clos_dic[k] for k,z in clos_dic.items()]

plt.figure(figsize=(10,10))

# make the scatter plot
plt.scatter(xd, yd, s=30, alpha=0.75, marker='o')

# determine best fit line
par = np.polyfit(xd, yd, 1, full=True)

slope=par[0][0]
intercept=par[0][1]
xl = [min(xd), max(xd)]
yl = [slope*xx + intercept  for xx in xl]

# coefficient of determination, plot text
variance = np.var(yd)
residuals = np.var([(slope*xx + intercept - yy)  for xx,yy in zip(xd,yd)])
Rsqr = np.round(1-residuals/variance, decimals=2)
plt.text(.7*max(xd)+.1*min(xd),0.25*max(yd)+.1*min(yd),'$R^2 = %0.2f$'% Rsqr, fontsize=30)

plt.ylabel("Weighted Sentiment",size=20)
plt.xlabel("Closeness",size=20)
plt.title('Weighted Sentiment vs. Closeness',size=25)

# error bounds
yerr = [abs(slope*xx + intercept - yy)  for xx,yy in zip(xd,yd)]
par = np.polyfit(xd, yerr, 2, full=True)
```

```
# yerrUpper = [(xx*slope+intercept)+(par[0][0]*xx**2 + par[0][1]*xx + par[0][2]) for xx,yy in zip(xd,yd)]
# yerrLower = [(xx*slope+intercept)-(par[0][0]*xx**2 + par[0][1]*xx + par[0][2]) for xx,yy in zip(xd,yd)]

for k,z in clos_dic.items():
  y,x = comm_dic2[k]['avg_sentiment'],clos_dic[k]
  if (abs(y) > 0.08) or k=='Dutch':
    plt.text(x+.005,y+.025, k, fontsize=12,fontweight='bold')

plt.plot(xl, yl, '-r')
plt.axhline(y=0, color='k', linestyle='-')
# plt.plot(xd, yerrLower, '--r')
# plt.plot(xd, yerrUpper, '--r')
plt.show()

xd = [(i-min(xd)) for i in xd]
xd_list = xd_list + xd
yd_list = yd_list + yd

load_dic = nx.load_centrality(G_cl,normalized=True)

yd = [comm_dic2[k]['avg_sentiment'] for k,z in load_dic.items()]
xd = [load_dic[k] for k,z in load_dic.items()]

plt.figure(figsize=(10,10))

# make the scatter plot
plt.scatter(xd, yd, s=30, alpha=0.75, marker='o')

# determine best fit line
par = np.polyfit(xd, yd, 1, full=True)

slope=par[0][0]
intercept=par[0][1]
xl = [min(xd), max(xd)]
yl = [slope*xx + intercept  for xx in xl]

# coefficient of determination, plot text
variance = np.var(yd)
residuals = np.var([(slope*xx + intercept - yy)  for xx,yy in zip(xd,yd)])
Rsqr = np.round(1-residuals/variance, decimals=2)
plt.text(.5*max(xd)+.1*min(xd),0.25*max(yd)+.1*min(yd),'$R^2 = %0.2f$'% Rsqr, fontsize=30)

plt.ylabel("Weighted Sentiment",size=20)
plt.xlabel("Load",size=20)
plt.title('Weighted Sentiment vs. Load',size=25)

# error bounds
yerr = [abs(slope*xx + intercept - yy)  for xx,yy in zip(xd,yd)]
par = np.polyfit(xd, yerr, 2, full=True)

# yerrUpper = [(xx*slope+intercept)+(par[0][0]*xx**2 + par[0][1]*xx + par[0][2]) for xx,yy in zip(xd,yd)]
# yerrLower = [(xx*slope+intercept)-(par[0][0]*xx**2 + par[0][1]*xx + par[0][2]) for xx,yy in zip(xd,yd)]

for k,z in load_dic.items():
  y,x = comm_dic2[k]['avg_sentiment'],load_dic[k]
  if (abs(y) > 0.08) and k!='French' or k=='Dutch':
    plt.text(x+0.003,y+.0025, k, fontsize=10,fontweight='bold')
  if (abs(y) > 0.08) and k=='French':
    plt.text(x+0.003,y-.015, k, fontsize=10,fontweight='bold')

plt.plot(xl, yl, '-r')
plt.axhline(y=0, color='k', linestyle='-')
# plt.plot(xd, yerrLower, '--r')
# plt.plot(xd, yerrUpper, '--r')
plt.show()
```

```
xd_list = xd_list + xd
yd_list = yd_list + yd

from matplotlib.pyplot import cm
def chunks(lst, n):
    for i in range(0, len(lst), n):
        yield lst[i:i + n]

xlst=xd_list
ylst=yd_list
n=int(len(xd_list)/3)
xd_list1 = [xlst[i:i + n] for i in range(0, len(xlst), n)]
yd_list1 = [ylst[i:i + n] for i in range(0, len(ylst), n)]

max_val = max([max(i) for i in xd_list1])

i = 0
while i < len(xd_list1):
  this_max = max(xd_list1[i])
  if this_max != max_val:
    xd_list1[i] = [j*max_val/this_max for j in xd_list1[i]]
  i += 1

plt.figure(figsize=(10,10))

arr = []
colors = cm.viridis_r(np.linspace(0, 1, len(yd_list1)))

j_ind = list(load_dic.keys()).index('Jew')

i = 0
s_sizes = [400,200,75]
for x,y,c in zip(xd_list1,yd_list1, colors):
  arr.append(plt.scatter(x, y, color=c,marker='o',s=s_sizes[i]))
  i += 1

xdj_list1 = [el[j_ind] for el in xd_list1]
ydj_list1 = [el[j_ind] for el in yd_list1]

i = 0
for x,y,c in zip(xdj_list1,ydj_list1, colors):
  arr.append(plt.scatter(x, y, color=c,marker='X',s=s_sizes[i]*4.5))
  i += 1

par = np.polyfit(xd, yd, 1, full=True)

slope=par[0][0]
intercept=par[0][1]
xl = [min(xd), max(xd)]
yl = [slope*xx + intercept  for xx in xl]

plt.ylabel("Weighted Sentiment",size=20)
plt.xlabel("Relative Centrality",size=20)
plt.title('Weighted Sentiment vs. Centrality',size=25)

yerr = [abs(slope*xx + intercept - yy)  for xx,yy in zip(xd,yd)]
par = np.polyfit(xd, yerr, 2, full=True)

plt.legend(arr,('Communicability','Closeness','Load'),prop={'size': 15})

plt.axhline(y=0, color='k', linestyle='-')

plt.show()

yd = [comm_dic2[k]['avg_sentiment'] for k,z in load_dic.items()]
```

```
xd = [load_dic[k] for k,z in load_dic.items()]

plt.figure(figsize=(10,10))

# make the scatter plot
plt.scatter(xd, yd, s=30, alpha=0.75, marker='o')

# determine best fit line
par = np.polyfit(xd, yd, 1, full=True)

slope=par[0][0]
intercept=par[0][1]
xl = [min(xd), max(xd)]
yl = [slope*xx + intercept  for xx in xl]

# coefficient of determination, plot text
variance = np.var(yd)
residuals = np.var([(slope*xx + intercept - yy)  for xx,yy in zip(xd,yd)])
Rsqr = np.round(1-residuals/variance, decimals=2)
plt.text(.5*max(xd)+.1*min(xd),0.25*max(yd)+.1*min(yd),'$R^2 = %0.2f$'% Rsqr, fontsize=30)

plt.ylabel("Weighted Sentiment",size=20)
plt.xlabel("Load",size=20)
plt.title('Weighted Sentiment vs. Load',size=25)

# error bounds
yerr = [abs(slope*xx + intercept - yy)  for xx,yy in zip(xd,yd)]
par = np.polyfit(xd, yerr, 2, full=True)

# yerrUpper = [(xx*slope+intercept)+(par[0][0]*xx**2 + par[0][1]*xx + par[0][2]) for xx,yy in zip(xd,yd)]
# yerrLower = [(xx*slope+intercept)-(par[0][0]*xx**2 + par[0][1]*xx + par[0][2]) for xx,yy in zip(xd,yd)]

for k,z in load_dic.items():
  y,x = comm_dic2[k]['avg_sentiment'],load_dic[k]
  if (abs(y) > 0.08) and k!='French' or k=='Dutch':
    plt.text(x+0.003,y+.0025, k, fontsize=10,fontweight='bold')
  if (abs(y) > 0.08) and k=='French':
    plt.text(x+0.003,y-.015, k, fontsize=10,fontweight='bold')

plt.plot(xl, yl, '-r')
plt.axhline(y=0, color='k', linestyle='-')
# plt.plot(xd, yerrLower, '--r')
# plt.plot(xd, yerrUpper, '--r')
plt.show()

yd = [comm_dic2[k]['avg_sentiment'] for k,z in load_dic.items()]
xd = [load_dic[k] for k,z in load_dic.items()]

plt.figure(figsize=(10,10))

# make the scatter plot
plt.scatter(xd, yd, s=30, alpha=0.75, marker='o')

# determine best fit line
par = np.polyfit(xd, yd, 1, full=True)

slope=par[0][0]
intercept=par[0][1]
xl = [min(xd), max(xd)]
yl = [slope*xx + intercept  for xx in xl]

# coefficient of determination, plot text
variance = np.var(yd)
residuals = np.var([(slope*xx + intercept - yy)  for xx,yy in zip(xd,yd)])
Rsqr = np.round(1-residuals/variance, decimals=2)
plt.text(.5*max(xd)+.1*min(xd),0.25*max(yd)+.1*min(yd),'$R^2 = %0.2f$'% Rsqr, fontsize=30)
```

```
plt.ylabel("Weighted Sentiment",size=20)
plt.xlabel("Load",size=20)
plt.title('Weighted Sentiment vs. Load',size=25)

# error bounds
yerr = [abs(slope*xx + intercept - yy)  for xx,yy in zip(xd,yd)]
par = np.polyfit(xd, yerr, 2, full=True)

# yerrUpper = [(xx*slope+intercept)+(par[0][0]*xx**2 + par[0][1]*xx + par[0][2]) for xx,yy in zip(xd,yd)]
# yerrLower = [(xx*slope+intercept)-(par[0][0]*xx**2 + par[0][1]*xx + par[0][2]) for xx,yy in zip(xd,yd)]

for k,z in load_dic.items():
  y,x = comm_dic2[k]['avg_sentiment'],load_dic[k]
  if (abs(y) > 0.08) and k!='French' or k=='Dutch':
    plt.text(x+0.003,y+.0025, k, fontsize=10,fontweight='bold')
  if (abs(y) > 0.08) and k=='French':
    plt.text(x+0.003,y-.015, k, fontsize=10,fontweight='bold')

plt.plot(xl, yl, '-r')
plt.axhline(y=0, color='k', linestyle='-')
# plt.plot(xd, yerrLower, '--r')
# plt.plot(xd, yerrUpper, '--r')
plt.show()
```

10b. Density

```
widths = [e_dic_cl[i]['width'] for i in G_cl.edges]
maxw = max(widths)
n = len(widths)
weighted_edge = round(sum(widths)/(n*(n-1)/2*maxw),5)
print('The Density of G is: {}'.format(round(nx.density(G_cl),3)))
print('The Weighted Density of G is: {}'.format(weighted_edge))
```

10c. Degree Histogram

```
def weight_attr(G_cl,maxw):
   weight={}
   for edge in G_cl.edges():
      ed=edge[0]
      de=edge[1]
      G_cl[ed][de]['weight']=random.randrange(1,maxw)
      weight[(ed,de)]=G_cl[ed][de]['weight']
   return weight

from pylab import hist

ds=dict(G_cl.degree(G_cl.nodes())).values()
bins=7

plt.figure(figsize=(20,7))
plt.subplot(121)
plt.xlabel("Degree");
hist(ds,bins=bins,color='g');
# histtype='step',
plt.ylabel("Number of nodes");
ss='Degree Histogram of G'
plt.title(ss,fontsize=20);
plt.subplot(122)
plt.xlabel("Degree");
hist(ds,bins=bins,color='g',log=True);
# histtype='step',
plt.ylabel("Number of nodes");
sss='Log Degree Histogram of G'
plt.title(sss,fontsize=20);
```

```
k = []
Pk = []

for node in list(G_cl.nodes()):
    degree = G_cl.degree(nbunch=node)
    try:
        post = k.index(degree)
    except ValueError as e:
        k.append(degree)
        Pk.append(1)
    else:
        Pk[post] += 1

logk=[]
logPk=[]
# get a double log representation
for i in range(len(k)):
    if k[i]>0:
        logk.append(math.log10(k[i]))
        logPk.append(math.log10(Pk[i]))

order = np.argsort(logk)
logk_array = np.array(logk)[order]
logPk_array = np.array(logPk)[order]
plt.plot(logk_array, logPk_array, ".")
m, c = np.polyfit(logk_array, logPk_array, 1)
print("m =",m, "c =",c)
plt.plot(logk_array, m*logk_array + c, "-");

if m>-3 and m<-2:
    print("The %s is scale-free" %name)
else:
    print("The %s is not scale-free" %name)

dsw=dict(G_cl.degree(G_cl.nodes()),weight='weight').values()
bins=7

plt.figure(figsize=(20,7))
plt.subplot(121)
plt.xlabel("Weighted Degree");
hist(dsw,bins=bins,color='g');
# histtype='step',
plt.ylabel("Number of nodes");
ss='Weighted Degree Histogram of G'
plt.title(ss,fontsize=20);
plt.subplot(122)
plt.xlabel("Weighted Degree");
hist(dsw,bins=bins,color='g',log=True);
# histtype='step',
plt.ylabel("Number of nodes");
sss='Log Weighted Degree Histogram of G'
plt.title(sss,fontsize=20);

k = []
Pk = []

for node in list(G_cl.nodes()):
    degree = G_cl.degree(nbunch=node,weight='weight')
    try:
        post = k.index(degree)
    except ValueError as e:
        k.append(degree)
        Pk.append(1)
    else:
        Pk[post] += 1
```

```
logk=[]
logPk=[]
# get a double log representation
for i in range(len(k)):
   if k[i]>0:
      logk.append(math.log10(k[i]))
      logPk.append(math.log10(Pk[i]))

order = np.argsort(logk)
logk_array = np.array(logk)[order]
logPk_array = np.array(logPk)[order]
plt.plot(logk_array, logPk_array, ".")
m, c = np.polyfit(logk_array, logPk_array, 1)
print("m =",m, "c =",c)
plt.plot(logk_array, m*logk_array + c, "-");

if m>-3 and m<-2:
   print("The wighted %s is scale-free" %name)
else:
   print("The weighted %s is not scale-free" %name)
```

10d. Connectivity

```
if nx.is_connected(G_cl)==True:
   print ("This graph is a connected graph")
else:
   print ("This graph is a disconnected graph and it has",nx.number_connected_components(sGw),"connected components" )
   giant = max(nx.connected_component_subgraphs(G_cl), key=len)
   sGwlcc=sGw.subgraph(giant)
   print ("The largest connected component of this graph is a weighted graph with %i nodes and %i edges" %(len(sGwlcc.nodes()),len(sGwlcc.ed
ges())))
   print ("The density of the largest connected component of this graph is %.3f" %nx.density(sGwlcc))
```

10e. Transitivity

```
print('The transitivity of G is: {}'.format(round(nx.transitivity(G_cl),3)))
```

10f. Clustering

```
Gt = nx.Graph()
wd = []
Gt.add_nodes_from(G_cl.nodes)
for edge in G_cl.edges:
  wd.append((edge[0],edge[1],e_dic_cl[edge]['width']))
Gt.add_weighted_edges_from(wd)
print('The weighted clustering coefficients are:')
clusdic = nx.clustering(Gt,weight='weight')
for z,v in clusdic.items():
  clusdic[z] = round(v,3)
clusdic

print('The average weighted clustering is: {}'.format(round(nx.average_clustering(Gt, nodes=None, weight='weight', count_zeros=True),3)))
```

10g. Distances

```
print('The diamaeter of G is: {}'.format(nx.diameter(Gt)))

print('The average shortest path of G is: {}'.format(round(nx.average_shortest_path_length(Gt),2)))
print('The weighted average shortest path of G is: {}'.format(round(nx.average_shortest_path_length(Gt, weight='weight'),2)))
```

10g. Assortativity

```
print('The weighted degree assortativity coefficient is: {}'.format(round(nx.degree_assortativity_coefficient(Gt, x='out', y='in', weight='weight',
nodes=None),2)))

print('The attribute assortativity coefficient is: {}'.format(get_att_assort(Gt,n_dic_cl)))
```

```
def make_mapping(G,n_dic,att='tag'):
  mapping = {}
  i = 0
  for node in G.nodes:
    if n_dic[node][att] not in mapping.keys():
      mapping[n_dic[node][att]] = i
      i += 1
  return mapping
nx.set_node_attributes(Gt, {z:n_dic_cl[z]['tag'] for z,v in n_dic.items()}, 'tag')
mapping = make_mapping(Gt,n_dic_cl,att='tag')
mix_mat = nx.attribute_mixing_matrix(Gt, 'tag',mapping=mapping)

from matplotlib import colors
print('Mixing matrix of G by NLP tag:')
df_mix = pd.DataFrame(data=mix_mat,index=mapping.keys(),columns=mapping.keys())*len(Gt.edges)*2
def background_gradient(s, m, M, cmap='PuBu', low=0, high=0):
    rng = M - m
    norm = colors.Normalize(m - (rng * low),
                            M + (rng * high))
    normed = norm(s.values)
    c = [colors.rgb2hex(x) for x in plt.cm.get_cmap(cmap)(normed)]
    return ['background-color: %s' % color for color in c]

df_mix.style.apply(background_gradient,
                   cmap='cool',
                   m=df_mix.min().min(),
                   M=df_mix.max().max(),
                   low=0,
                   high=0.1)
```