Decomplexifying the network pipeline: a tool for RDF/Wikidata to network analysis

Julie M. Birkholz[1] and Albert Meroño-Peñuela[2]

[1]WeChangEd, Department of Literary Studies, Ghent University, Ghent, Belgium
[2]Department of Computer Science, Vrije Universiteit Amsterdam, The Netherlands

Knowledge Graphs that use the Resource Description Framework language (RDF) as a knowledge representation paradigm are increasingly popular in Digital Humanities, and represent a valuable source of data for network analysis. However, digital scholars interested in network approaches over RDF graphs have to deal with complex workflows and frameworks in order to perform their analyses. These complexities exacerbate complications in reproducing and replicating their work. In this paper, we detail a proof of concept to combine popular libraries in RDF data management and network analysis in one single, publicly accessible Jupyter Notebook that enables a structured approach to network analyses of RDF graphs. What sets our work apart specifically is its flexibility in quickly re-running network analyses over slightly modified RDF graphs, and ensuring transparency in making the code visible. We explain this approach through two case studies: women editors in Europe in the 19th century, and provenance of the harmonization of the historical Dutch censuses (1795-1971). This approach affords the researcher to quickly, easily, efficiently and with increased reliability project and analyse networks from RDF.

1 Introduction

Linked Data is an increasingly common way to publish structured data in the Humanities (de Boer et al., 2014, Meroño-Peñuela et al., 2015, Thornton et al., 2017). As Tim Berners-Lee, the "creator" of the Semantic Web, described - Linked Data “provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries.” ((W3C), 2011). Thus facilitating accessibility of knowledge on historical and cultural objects in a format readable by both humans and machines. For example, through standards such as the Resource Description Framework (RDF), natural language statements such as ‘‘`George Orwell wrote`

1984'' can be expressed as a triple consisting of: a subject (`:George_Orwell`), a predicate (`:wrote`), and an object (`:1984`). This knowledge can be retrieved by machines through a unique and global identifier (Uniform Resource Identifiers - URIs). This affords a networked archive, bringing together publicly available materials distributed in libraries, archives and museums; and thus allowing the researcher to integrate, and implement an unprecedented amount of often unstructured, siloed data, in lightning speed. Such an ontology or data model affords access, merging of information, and enrichment through efficiently linking of information on objects, entities and relations of collections to other collections.

Technically speaking data represented in the RDF language is structurally a graph. Thus it inherently allows us to infer relations, bundling any common affiliation between objects and attributes. From a research point of view this has led to a tendency to study RDF as a network. The study of networks and specifically the study of social networks has its roots in sociological theories where relationships form a part of the basis for understanding behavior(Durkheim, 1951, Simmel, 1955) where all actions are embedded in networks.(Granovetter, 1985) These relations – a set of edges, between nodes (entities) – define a network. These social networks reflect types of relations (e.g., a friendship tie in a friendship network or advice tie in an advice network). The study of networks, and in particular social networks, have been and are on the rise, providing explanations for relational and systematic phenomena(Borgatti and Foster, 2003), as it moves beyond explanations based on individual factors. For example not that someone's age explains their success, but rather the structure of their social network.(Granovetter, 1985)

The identification of networks is often thought of as a laborious task. It is traditionally done in many fields by searching through archival sources to identify nodes and edges, and reshaping data that is often not collected as relational, but from which one can infer relations. This entails integrating, and implementing a large amount of often unstructured, siloed and incomplete data to reconstruct relations between nodes and edges. Thus information about relations where a social network can be inferred from RDF provides a great advantage for exploring social networks embedded in this data. Networks can efficiently be reconstructed with the development of specific SPARQL queries to reflect different lenses of relations. For example, generating networks of different time periods, of different types of relations, with different boundaries (looking at relations of one city versus one country, or a neighborhood to a street) over the same data source.

Modeling data as networks affords the implementation of network analysis. Network analysis –the method used to analyze relations– provides a lens to investigate these diverse complex relational dynamics to examine structure, content or function. For social networks, which are the focus of the examples we provide in this paper, the structure of networks and positions of actors in these structures are seen as proxies for understanding social structure (Burt, 1980, Coleman, 1988).

The analysis of networks from RDF is largely done with a pipeline of tools (i.e. (Gil and Groth, 2011, Groth and Gil, 2011)). This starts with a data source, and the tools necessary for querying the specific data and relations. For example one workflow may be: the Wikidata Query Service, which allows one to query linked data in the Wikisphere through a SPARQL query, and can be exported in a number of formats; or the data might be stored in a database and is extract-able as a JSON(-LD) file. Moving from these file types, this relational data needs to converted into a file type that is readable by a network analysis software. Typical network analysis software use a range of inputs depending on the program. The two most commonly used user friendly network analysis and visualization programs with a graphic user interface are Gephi[1] and UCINet[2]. These programs allow the implementation of various types of input files; for example: .csvs, matrices and DL files, as well as program specific files. Then pending the required analysis there are a number of export options to further reuse these results as data. This, for example, could include analysing network measures and considering them as a variable in a statistical model in a program such as SPSS, or R. Thus the current workflow approaches for working with network data from RDF requires researchers to work through multiple programs to specify queries, extract networks and export data as matrices, and implement network analysis tools to investigate graphs.

In addition, in building such a pipeline we lose sight of the hermeneutics of the research objects.(Gibbs and Owens, 2013) Researchers are often faced with black boxed tools that limit their understanding of the projection, generation, analysis or reformatting that occurs with each step. With each use of an additional program, algorithm or command, the data gets re-"massaged" and shaped. This further becomes an issue, when the development of such a pipeline is a technical adversary for domain experts (e.g. historians, literary scholars) with (traditionally) limited technical knowledge; but also for researchers with specific expertise in RDF or networks. Thus, we argue there is a need, within the DH community, to reduce this RDF-to-network analysis pipeline without creating another domain or research question specific tool, and while maintaining oversight over the process from RDF-to graph-to network analysis.

To address these issues, we propose the use of a Jupyter notebook that integrates the Python packages: RDFLib[3] with NetworkX[4].[5] This results in a reusable workflow that allows network analyses over RDF data to be more accessible, flexible, transparent and iterative. This is due to that increases the reliability in exploring all the possible social networks within the available RDF, as well as increases the speed, ease, and efficiency of the necessary steps of RDF to network analysis. What specifically sets our work apart from previous workflows is its flexibility in quickly re-running network analyses over slightly modified RDF graphs, while maintaining the code visible for transparency and learning. We outline this pipeline through two case studies:

1. a case of the social networks of 19th century women editors in Europe available on Wikidata, and
2. provenance of the harmonization of the historical Dutch censuses (1795-1971)

to explain how it can be useful for humanities research.

2 Method

The notebook consists of five "cells", which are actionable code blocks, shown here in Figure 1. The output of all these processes can be selected and copy-pasted for further

[1] https://gephi.org/

[2] https://sites.google.com/site/ucinetsoftware/home

[3] https://github.com/RDFLib/RDFLib

[4] https://NetworkX.github.io/

[5] The full notebook is available at https://github.com/descepolo/rdf-network-analysis/blob/master/rdf-network-analysis.ipynb. A Google Colaboratory version of the notebook is also available, which makes it executable on the web with no need of local installation: https://colab.research.google.com/github/descepolo/rdf-network-analysis/blob/master/rdf-network-analysis.ipynb.

reuse in graph processing frameworks or directly in reports or papers.

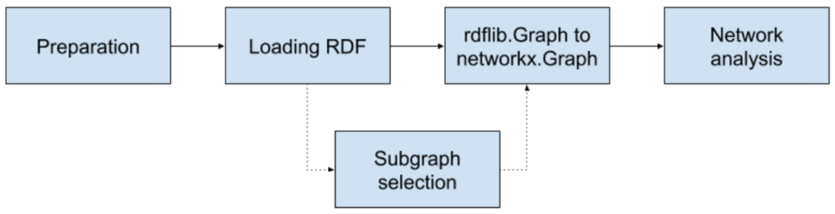


Figure 1: Workflow of the RDF Network Analysis Jupyter notebook

2.1 Preparation

As a first step the notebook loads the relevant packages - RDFLib and NetworkX. RDFLib is a Python package for working with RDF that includes parsers and serializers for RDF/XML, N3, NTriples, N-Quads, Turtle, TriX, RDFa and Microdata; a graph interface; store implementations for in memory storage and persistent storage on top of the Berkeley DB; and a SPARQL 1.1 implementation (Krech, 2006). This facilitates a flexible environment for loading and manipulating RDF graphs. Then the user is prompted to input the full path to an RDF graph to load the RDF graphs. This can be any local or online RDF file.

2.2 Subgraph Selection

Users select a specific network in the RDF graph. The efficient aggregation of different snapshots of the networks can be achieved through a SPARQL query. SPARQL is a Semantic Web query language for databases which enable the ability to retrieve and manipulate data RDF specifically (Segaran et al., 2009).

2.3 From RDFLib to NetworkX

In order to generate a network, this RDF needs to converted into a matrix. This is accomplished through a conversion of `RDFLib.Graph` to `NetworkX.Graph`. This prepares a file of the identified graph for analysis in NetworkX.

The Python library NetworkX enables the analysis of networks of around 10 million nodes and 100 million edges.(Hagberg and Conway, 2010) It is ideal for use for digital humanities as it affords the use of many types of networks, including directed graphs, and graphs with and self loops; while not maintaining strict object functions.(Hagberg et al., 2008) This implies that in the case of RDF which may have many and multiple types of networks embedded in the triples it will model anything that is structured as a matrices. This could include networks that we do not discuss here in this paper such as affiliation or two-mode networks, semantic networks and so forth. Thus the tool, which operates in the more general space of RDF models, does not limit the boundaries of inspection by imposing specific network models, leaving this choice to the user.

2.4 Network Analysis

Networks can be represented as graphs where positions and structures are systematically analyzed.(Wasserman et al., 1994) These principles originate from graph theory, which provides mathematical descriptions of characteristics.(Van Steen, 2010)

The networks can then be analyzed in NetworkX considering a number of characteristics of the network, as well as statistical analyses, see Table 1. Proposed Network Characteristics. We have selected a standard, non-exhaustive, set of one-mode complete network measures. This is to establish the proof of concept, of course in practice any network measure that is included in NetworkX could be implement in this notebook, for example measures of community detection, to other measures of centrality.[6] For a more exhaustive list and explanation of network measures see (Wasserman et al., 1994).

Following this selection the network analysis is run and the results are printed, as well as a basic visualization which serves for the researcher to confirm a first accuracy check of the network, e.g. were the correct node and edges selected?; does something look strange or potentially missed in the query?, that can now be amended.

Network Concepts	Network measures
network size	total number of nodes, and the average number of edges
power centrality	nodal position: e.g. degree centrality, betweenness, and eigenvector centrality (Freeman, 1978)
density	a value of the proportion of all possible ties that are present

Table 1: Network Characteristics.

3 Case Studies

In this Section we validate our approach using two different case studies for the Digital Humanities: the social networks of women editors in Europe in the 19th century; and the provenance graphs of harmonization transformations performed in the Dutch historical censuses. The use of these cases are to demonstrate the use of the notebook, not a network study with elaborated research questions and operationalized network measures.

3.1 Women Editors in Europe in the 19th century

The 19th century in Europe, was one of the onset and rise of industrialization, altered the socioeconomic and cultural norms influencing the movement of people through advancements in train infrastructure and technologies in food and consumer goods, and investments in education throughout Europe. This also led to an increasing advancement of women's rights and positions in society. The ERC "Agents of Change: Women Editors and Socio-Cultural Transformation in Europe, 1710-1920" (acronym WeChangEd) directed by Marianne Van Remoortel and based at the Department of Literary Studies, Ghent University, Belgium (project Agents of Change: Women Editors and Socio-Cultural Transformation in Europe, 2015), questioned how the press and periodical editorship in particular enabled women to take a prominent role in public

[6] It is not the goal of this paper to explain the operationalization of theoretical concepts to network measures, but it should be considered by humanities researchers in deciding on applicable measures to include in their research.

life, to influence public opinion and to shape transnational processes of change. To facilitate the collection of biographical records, and archival evidence of women editors in Europe a Linked Data model was developed (Schelstraete and Van Remoortel, 2019). This model afforded the cataloguing and tracing of different social networks in which the women participated.

This resulted in a large and growing database which includes 1700+ persons, 1600+ periodicals and 200+ organizations, as well as biographical information of these entities and relations between them, as identified through archival research. This data is available as the WCD Database, as subsets of data stored as .csv (Van Remoortel et al., 2020). In April 2020, the WCD database was imported to Wikidata (Thornton et al., ming) to facilitate the reuse and integration of this information with other Linked Open Data sources. The WeChangEd data can be identified in Wikidata through the unique property instance of WeChangEd ID P7947, see - `https://www.wikidata.org/wiki/Property:P7947`. This resulted in 3661 instances of data which compromises people, periodicals, organizations, as well as records of the relationships between these three entities, biographical information about these instances, and so forth. The complete dataset can be found via a Wikidata Query Service via - `https://w.wiki/QiQ`.

Identifying historical social networks is a laborious task, thus having the information on relations in Wikidata, and specifically as RDF, allows the researcher to explore historical social networks of the past in a more valid and flexible manner. The validity is increased, as the information is shared with the community, where it can be cross-checked, questioned, and enriched through the edit functions of Wikidata. As we show here through this example, the flexibility is affording by this pipeline.

In exploring how a researcher can identify social networks of these editors we display here three examples of projecting personal relationships of female editors between individuals as identified within the WeChangEd dataset. To identify these relationships we developed three SPARQL queries for the Wikidata Query Service, which we detail here below, and are also available at: `https://w.wiki/Qtr`, `https://w.wiki/QiQ`, and `https://w.wiki/QcS`, respectively. Using these graphs as input for the method described in Section 2, we convert these graphs to a NetworkX file and the network analysis is executed. We implement this query in the notebook resulting in three different network projections, and reflect on the implications for digital humanities researchers in compiling social networks from the past.

The first network represents a query on the entire WCD dataset, to identify kinship relations, this includes any identified siblings, parents, unmarried partner, spouse, or children of female editors `https://w.wiki/Qtr` (see Listing 1).

This results in a network of all female editors and their relationships as identified in Wikidata, where nodes are individuals and edges or ties of represent a personal relationship, see Figure 2.

```
SELECT DISTINCT ?item ?o ?itemLabel ?sibling ?spouse ?partner ?father
?mother ?child
WHERE

{
# find occupation editors
?item wdt:P106 wd:Q1607826.
?item wdt:P7947 ?o .

# that are female
?item wdt:P21 wd:Q6581072.

# that have a birth and death date
?item wdt:P569 ?birthDate.
?item wdt:P570 ?deathDate .

# with kinship: sibling
OPTIONAL { ?item wdt:P3373 ?sibling .}
# with kinship: spouse
OPTIONAL { ?item wdt:P26 ?spouse .}
# with kinship: unmarried partner
OPTIONAL { ?item wdt:P451 ?partner .}

# with kinship: father
OPTIONAL { ?item wdt:P22 ?father .}
# with kinship: mother
OPTIONAL { ?item wdt:P25 ?mother .}
# with kinship: child
OPTIONAL { ?item wdt:P40 ?child .}

# labels
SERVICE wikibase:label { bd:serviceParam wikibase:language
"[AUTO_LANGUAGE],en". }

} ORDER BY ?birthDate ?deathDate
```

Listing 1: SPARQL query for all female authors and their kinship relations

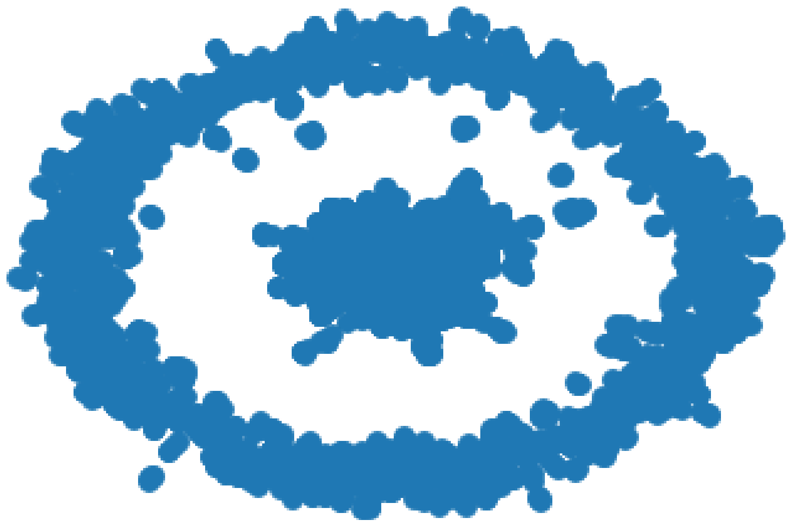


Figure 2: Network of editors

```
SELECT DISTINCT ?item ?o ?itemLabel ?sibling ?spouse ?partner ?father
?mother ?child
WHERE

{
# find occupation editors
?item wdt:P106 wd:Q1607826.
?item wdt:P7947 ?o .

# that are female
?item wdt:P21 wd:Q6581072.

# that have a birth and death date
?item wdt:P569 ?birthDate.
?item wdt:P570 ?deathDate.

 # that is British
?item wdt:P27 wd:Q174193.

# with kinship: sibling
OPTIONAL { ?item wdt:P3373 ?sibling .}
# with kinship: spouse
OPTIONAL { ?item wdt:P26 ?spouse .}
# with kinship: unmarried partner
OPTIONAL { ?item wdt:P451 ?partner .}

# with kinship: father
OPTIONAL { ?item wdt:P22 ?father .}
# with kinship: mother
OPTIONAL { ?item wdt:P25 ?mother .}
# with kinship: child
OPTIONAL { ?item wdt:P40 ?child .}

# only active in the 19th century
FILTER ( ?birthDate >= "1800-01-01T00:00:00Z"^^xsd:dateTime &&
?deathDate <= "1898-12-31T00:00:00Z"^^xsd:dateTime )

# labels
SERVICE wikibase:label { bd:serviceParam wikibase:language
"[AUTO_LANGUAGE],en". }

} ORDER BY ?birthDate ?deathDate
```

Listing 2: SPARQL query for relationships of British female editors of periodicals in the 19th century in Wikidata

In this second selection we aim to show, how to refine the query, to select a more bounded set of nodes. This is a bounded selection of relations from within the WCD dataset but specifically of 19th century British female editors and their kinship relations, this includes any identified siblings, parents, unmarried partner, spouse, or children: `https://w.wiki/QnA` (see Listing 2).

This results in a network of the personal relations of 19th century British female editors, where nodes are individuals and edges are relationships, see Figure 3. This network is a subset of the larger graph, but with parameters of time - editors living during the 19th century, and place -what was then the United Kingdom of Great Britain and Ireland.

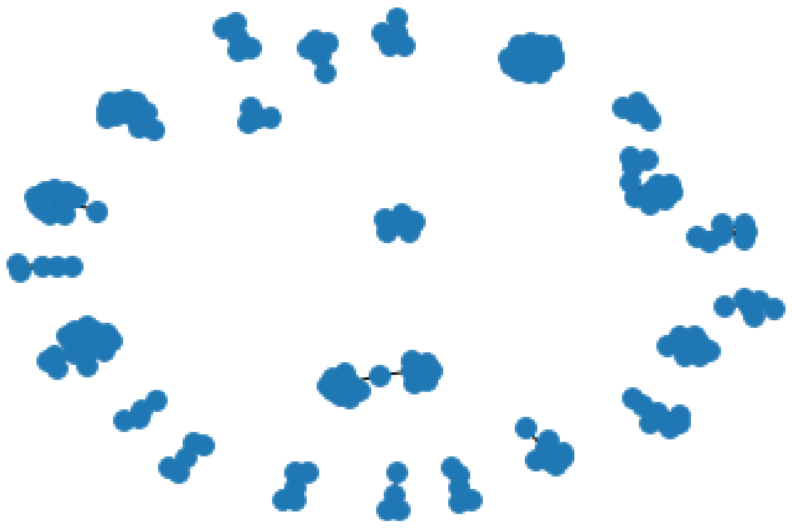


Figure 3: Network of 19th Century British female editors

The third case, aims to represent a different subset of the data, that is looking at relationships between editors based on language, instead of a geographical or political space. This selection represents a query of 19th century German speaking female editors and their kinship relations, this includes any identified siblings, parents, unmarried partner, spouse, or children:- `https://w.wiki/QnB` (see Listing 3).

This results in a network of relations of German speaking female editors as identified on Wikidata. Selecting German speaking instead of a specific empires and or nation-state provides a broader query for identifying possible interactions between the German-speaking community in the 19th century. This results in a network of individuals as nodes and edges as relations, see Figure 4.

The complete results for two specific social networks of 19th century British female editors and 19th century German speaking female editors can be found in detail in the appendix. The results show the network analysis on connected components or groups of connected individuals, most central nodes, and communities. A researcher can use these results, combined with the visualizations to further explore these relations either returning to archival materials to investigate previously understudied relations, or further analyse the structure and positions within these network to explain social capital of the periodicals the editors edited or kinship relations.

These three examples from within the WCD dataset on Wikidata display the flexibility of this approach in moving through a dataset, to generate social networks. This notebook, in contrast to other workflows allows researchers to consider aspects of space, time, place and other parameters of the data within a few steps and seconds; where the researcher can move and back and forth between the raw data, the query, the network projection, and the analysis, to compile the most suitable, reliable graph from the available data. It serves as both an efficient approach to explore the social relations within a dataset, as well as to validly and reliably generate a social network and conduct social network analysis of the networks from RDF.

```
SELECT DISTINCT ?item ?o ?itemLabel  ?sibling ?spouse ?partner ?father
?mother ?child
WHERE

{
# find occupation editors
?item wdt:P106 wd:Q1607826.
?item wdt:P7947 ?o .

# that are female
?item wdt:P21 wd:Q6581072.

# that have a birth and death date
?item wdt:P569 ?birthDate.
?item wdt:P570 ?deathDate.

# that speaks German
?item wdt:P1412 wd:Q188.

# with kinship: sibling
OPTIONAL { ?item wdt:P3373 ?sibling .}
# with kinship: spouse
OPTIONAL { ?item wdt:P26 ?spouse .}

# with kinship: father
OPTIONAL { ?item wdt:P22 ?father .}
# with kinship: mother
OPTIONAL { ?item wdt:P25 ?mother .}
# with kinship: child
OPTIONAL { ?item wdt:P40 ?child .}

# only active in the 19th century
FILTER ( ?birthDate >= "1800-01-01T00:00:00Z"^^xsd:dateTime &&
?deathDate <= "1898-12-31T00:00:00Z"^^xsd:dateTime )

# labels
SERVICE wikibase:label { bd:serviceParam wikibase:language
"[AUTO_LANGUAGE],en". }

} ORDER BY ?birthDate ?deathDate
```

Listing 3: SPARQL query for relationships of German speaking editors of periodicals in the 19th century in Wikidata

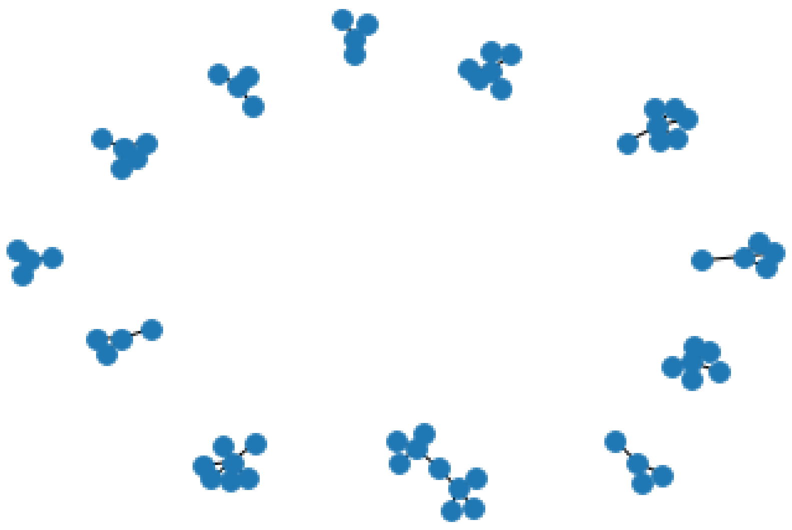


Figure 4: Network of 19th Century German speaking female editors

3.2 CEDAR: Harmonization Provenance of the Dutch Historical Censuses (1795-1971)

The Dutch historical censuses were collected in the Netherlands in the period 1795–1971, in 17 different editions, once every 10 years. The government counted all the country's population, door-to-door, and aggregated the results in three different census types: demographic (age, gender, marital status, location, belief), occupational (occupation, occupation segment, position within the occupation), and housing (ships, private houses, government buildings, occupied status). After 1971, this exhaustive collection stopped due to social opposition, and the government switched to municipal registers and sampling (Ashkpour et al., 2015). Various projects have digitized the resulting census data (CBS; IISH; Data Archiving and Networked Services[7], DANS; and the Netherlands Interdisciplinary Demographic Institute[8], NIDI), and have manually translated them into a collection of 507 Excel spreadsheets and 2,288 census tables.[9]. The CEDAR project[10] takes these spreadsheets as input, and produces a Knowledge Graph of 6.8 million statistical observations (Meroño-Peñuela et al., 2015) many of which went through an harmonization process to satisfy the standardization needs of historians for their querying (Ashkpour et al., 2015).

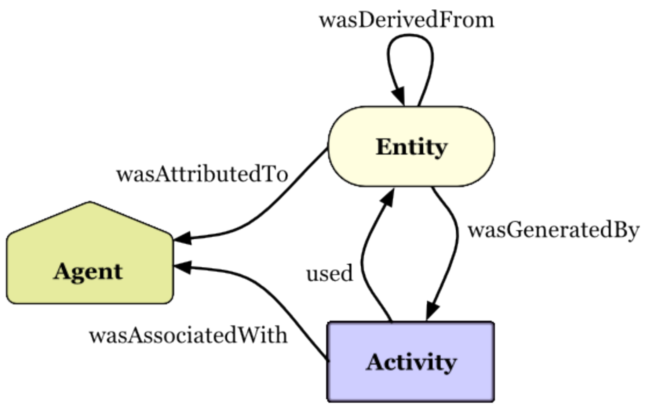


Figure 5: Provenance model of W3C PROV (Lebo et al., 2013).

In this case study, we use the CEDAR Knowledge Graph (Meroño-Peñuela et al., 2015) with our proposed approach to explain how to a researcher can consider network similarities and differences between various historical census data points and their *provenance* information. Historians are particularly interested in the transformation and manipulations that occurred in this harmonization process in generating these data points; as this signals their correctness and hence its reliability. Fortunately, the CEDAR Knowledge Graph documents the harmonization transformations of all data points using the W3C PROV standard (Lebo et al., 2013). This standard models provenance as the interactions between various *entities* (the objects subject to transformations, i.e. the census data points), *activities* (the transformation processes themselves, i.e. the harmonization rules) and *agents* (the persons or programs commanding the transformations) as shown in Figure 5.

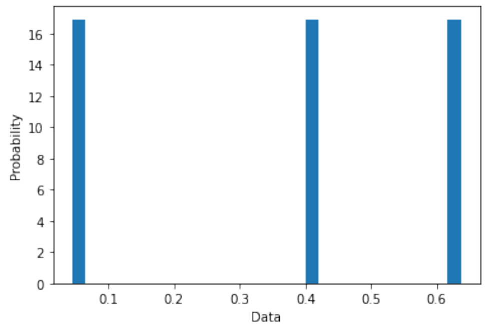
We select two arbitrary observations of the census, `VT_1859_01_H1-S8-J647-h` (observation 1, o_1) and `VT_1920_01_T-S0-R10108-h` (observation 2, o_2), and their corre-

[7] See `http://www.dans.knaw.nl/`
[8] `http://www.nidi.knaw.nl/en/`
[9] `http://volkstellingen.nl/`
[10] `https://www.cedar-project.nl/`

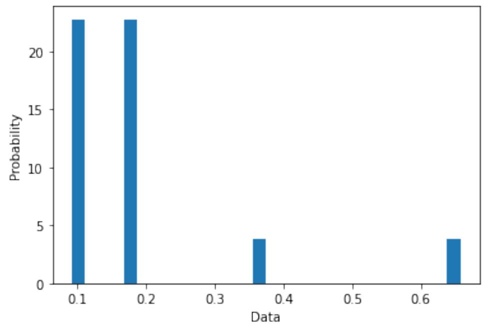
```
CONSTRUCT {
 ?obs ?obs_p ?obs_o .
 ?act ?act_p ?act_o .
} WHERE {
 VALUES ?obs {:VT_1859_01_H1-S8-J647-h :VT_1920_01_T-S0-R10108-h}
 ?obs prov:wasGeneratedBy ?act .
 ?obs ?obs_p ?obs_o .
 ?act ?act_p ?act_o .
}
```

Listing 4: SPARQL query for the harmonization provenance graphs of two census observations.

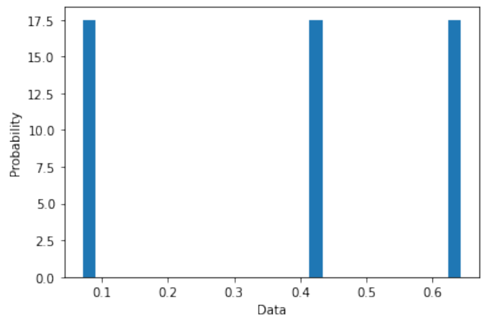
sponding provenance traces with the query shown in Listing 4 against the CEDAR SPARQL endpoint[11]. We use the graphs returned by this query as input for the notebook.[12]



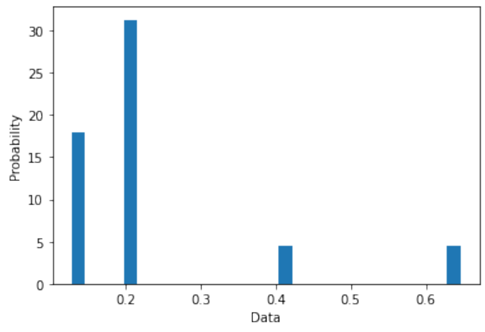
(a) o_1 degree centrality.



(b) o_1 eigenvector centrality.



(c) o_2 degree centrality.



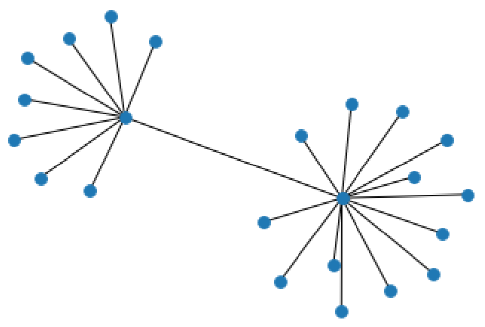
(d) o_2 eigenvector centrality.

Figure 6: Histogram plots of o_1 and o_2 degree and eigenvector centrality.

We use the provenance graphs of o_1 and o_2 as input for the method described in Section 2. We execute the preparation block; we use the query of Listing 4 as subgraph selection; we execute the network conversion block; and finally we execute the network analysis block. The output networks as plotted by the notebook are shown in Figure 7. We can observe that for both cases the network is 2-star shaped, with the nodes representing the observation and the activity at the center of these stars and various nodes describing their properties, as expected. One edge (`prov:wasGeneratedBy`) connects these two nodes. An noticeable difference is that while o_1 (Figure 7a) is

[11] `https://api.druid.datalegend.net/datasets/datalegend/CEDAR-S/services/CEDAR-S/sparql`

[12] The input graphs are also available at `https://github.com/albertmeronyo/rdf-network-analysis/blob/master/uc1.nt` and `https://github.com/albertmeronyo/rdf-network-analysis/blob/master/uc2.nt`


(a) Network of o_1

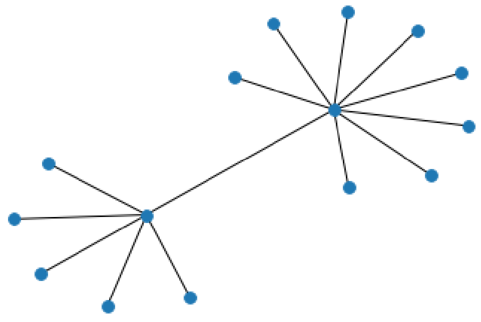

(b) Network of o_2

Figure 7: Network plots of two CEDAR observations and their harmonization provenance traces.

transformed by 6 different harmonization rules, o_2 (Figure 7b) is only affected by 3. This can provide interesting insights for historians, who may be keen to examine statistical observations that have been subject to a higher number of transformations (and therefore more prone to errors) and the relations of these transformations to their immediate context. In this sense, visualizing these network contexts can be a powerful tool for interpretation.

Additionally, Figure 6 shows the histograms for degree and eigenvector centrality drawn by the notebook for both graphs. This is a more aggregated view on the networks, showing similar behaviour for o_1 and o_2 (due to the structural similarity of provenance graphs) but also interesting differences. For example, o_1 eigenvector centrality shows a more normal distribution due to the higher variety of node influence in a more varied network. The remaining network statistics can be found upon the execution of these two examples in the notebook at `https://github.com/descepolo/rdf-network-analysis/blob/master/rdf-network-analysis.ipynb`.

4 Conclusion and Future Work

In this paper we have detailed how we have proposed to combine popular libraries in RDF data management and network analysis in one single, publicly accessible Jupyter Notebook that enables a structured approach to network analyses of RDF graphs. With the proposed Juypter notebook we have developed a transparent and iterative tool for RDF to network in research. The open code and user-friendliness of the notebook ensures flexibility for users in implementing different aspects of the two libraries that we did address here in this demonstration. In addition, we have demonstrated through the tool and use cases how this affords the reuse and accessibly for non-technical scholars of RDF, as well as increase the efficiency and flexibility of use for generating networks from RDF. This approach facilitates the study of diverse types of networks from RDF and thus study of relational phenomenon in the Humanities and beyond.

In addition, this approach proves, contrary to the trend in the digital humanities, that we do not need a new network software that converts diverse file types to make fundamental improvements on both the quality of the networks used in research, as well as the the analysis of networks. Rather, as we presented, a fundamental rethinking of how data on social networks is structured, manipulated and pushed through a pipeline is needed to efficiently generate, project and evaluate networks.

This approach increases the flexibility, compared to traditional network workflows- where the analyst would prepare a matrix for each projection of a network, go back to source material every time to reshape the data and networks based on different periods, or parameters (e.g. variables such as country of birth, gender, language of entities), and push it through the workflow. Such an approach reduces the technical adversary of knowledge on RDF and network analysis, while avoiding a black boxed software, as well as retains a hermeneutic approach to the source data, allowing the researcher to iteratively and efficiently requery, reshape and reanalyze the networks embedded in RDF.

Acknowledgements

The authors would like to thank Prof. dr. Paul Groth for his feedback on this work but also for his ongoing support of all of our crazy ideas. The first author would like to thank her partner- Radboud Reijn for being so patient, and secondly the continued support from Prof. dr. Marianne Van Remoortel in proposing and believing in a DH project for periodical studies. The work of the first author was supported by the H2020 European Research Council under the ERC Starting Grant agreement no. 639668. This work has been partly supported by the Dutch national project CLARIAH.

References

Ashkpour, A., A. Meroño-Peñuela, and K. Mandemakers
2015. The aggregate Dutch historical censuses: Harmonization and RDF. *Historical Methods: A Journal of Quantitative and Interdisciplinary History*, 48(4):230–245.

Borgatti, S. P. and P. C. Foster
2003. The network paradigm in organizational research: A review and typology. *Journal of management*, 29(6):991–1013.

Burt, R. S.
1980. Models of network structure. *Annual review of sociology*, 6(1):79–141.

Coleman, J. S.
1988. Social capital in the creation of human capital. *American journal of sociology*, 94:S95–S120.

de Boer, V., M. van Rossum, J. Leinenga, and R. Hoekstra
2014. Dutch ships and sailors linked data. In *International Semantic Web Conference*, Pp. 229–244. Springer.

Durkheim, E.
1951. Suicide: A study in sociology (ja spaulding & g. simpson, trans.). *Glencoe, IL: Free Press.(Original work published 1897)*.

Freeman, L. C.
1978. Centrality in social networks conceptual clarification. *Social networks*, 1(3):215–239.

Gibbs, F. and T. Owens
2013. The hermeneutics of data and historical writing. *Writing history in the digital age*, 159.

Gil, Y. and P. Groth
2011. LinkedDataLens: linked data as a network of networks. In *Proceedings of the sixth international conference on Knowledge capture*, Pp. 191–192. ACM.

Granovetter, M.
1985. Economic action and social structure: The problem of embeddedness. *American journal of sociology*, 91(3):481–510.

Groth, P. and Y. Gil
2011. Linked data for network science. In *Proceedings of the First International Conference on Linked Science-Volume 783*, Pp. 1–12. CEUR-WS. org.

Hagberg, A. and D. Conway
2010. Hacking social networks using the python programming language. *Sunbelt 2010, Riva del Garda, Italy*.

Hagberg, A., P. Swart, and D. S Chult
2008. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States).

Krech, D.
2006. Rdflib: A python library for working with rdf.

Lebo, T., S. Sahoo, D. McGuinness, K. Belhajjame, J. Cheney, D. Corsar, D. Garijo, S. Soiland-Reyes, S. Zednik, and J. Zhao
2013. Prov-o: The prov ontology. *W3C recommendation*.

Meroño-Peñuela, A., A. Ashkpour, C. Guéret, and S. Schlobach
2015. CEDAR: the Dutch Historical Censuses as Linked Open Data. *Semantic Web*, 8(2):297–310.

project Agents of Change: Women Editors, E. and .-. a. W. Socio-Cultural Transformation in Europe
2015. Wechanged.

Schelstraete, J. and M. Van Remoortel
2019. Towards a sustainable and collaborative data model for periodical studies. *Media History*, 25(3):336–354.

Segaran, T., C. Evans, and J. Taylor
2009. *Programming the Semantic Web: Build Flexible Applications with Graph Data*. " O'Reilly Media, Inc.".

Simmel, G.
1955. The web of group-affiliations. conflict and the web of groupaffiliations.

Thornton, K., J. M. Birkholz, M. V. Remoortel, and S.-N. Kenneth
forthcoming. Working paper on bringing to light to women editors of the past through linked open data on wikidata.

Thornton, K., E. Cochrane, T. Ledoux, B. Caron, and C. Wilson
2017. Modeling the domain of digital preservation in wikidata. In *Proceedings of ACM International Conference on Digital Preservation, Kyoto, Japan*.

Van Remoortel, M., J. Birkholz, J. Schelstrate, M. Alesina, C. Bezari, C. D'Eer, E. Forestier, N. De Grave-Geeraert, M. Goethals, J. A., and G. Vanhulle
2020. Wechanged database.

Van Steen, M.
2010. Graph theory and complex networks. *An introduction*, 144.

(W3C), W. W. W. C.
2011. W3c semantic web activity.

Wasserman, S., K. Faust, et al.
1994. *Social network analysis: Methods and applications*, volume 8. Cambridge university press.

A. 19th century British female editors and their kinship relations as present in Wikidata

May 15, 2020

1 Network Analysis of RDF Graphs

In this notebook we provide basic facilities for performing network analyses of RDF graphs easily with Python rdflib and networkx

We do this in 4 steps: 1. Load an arbitrary RDF graph into rdflib 2. Get a subgraph of relevance (optional) 3. Convert the rdflib Graph into an networkx Graph, as shown here 4. Get an network analysis report by running networkx's algorithms on that data structure

[13]:

1.1 0. Preparation

```
# Install required packages in the current Jupyter kernel
# Uncomment the following lines if you need to install these libraries
# If you run into permission issues, try with the --user option
import sys
# !pip install -q rdflib networkx matplotlib scipy
!{sys.executable} -m pip install rdflib networkx matplotlib scipy --user

# Imports
from rdflib import Graph as RDFGraph
from rdflib.extras.external_graph_libs import rdflib_to_networkx_graph
import networkx as nx
from networkx import Graph as NXGraph
import matplotlib.pyplot as plt
import statistics
import collections
```

```
Requirement already satisfied: rdflib in /home/amp/.local/lib/python3.8/site-
packages (5.0.0)
Requirement already satisfied: networkx in /home/amp/.local/lib/python3.8/site-
packages (2.4)
Requirement already satisfied: matplotlib in
/home/amp/.local/lib/python3.8/site-packages (3.2.1)
Requirement already satisfied: scipy in /home/amp/.local/lib/python3.8/site-
packages (1.4.1)
Requirement already satisfied: pyparsing in /usr/lib/python3/dist-packages (from
rdflib) (2.4.6)
Requirement already satisfied: six in /usr/lib/python3/dist-packages (from rdflib)
(1.14.0)
```

```
Requirement already satisfied: isodate in /home/amp/.local/lib/python3.8/site-
packages (from rdflib) (0.6.0)
Requirement already satisfied: decorator>=4.3.0 in /usr/lib/python3/dist-
packages (from networkx) (4.4.2)
Requirement already satisfied: kiwisolver>=1.0.1 in
/home/amp/.local/lib/python3.8/site-packages (from matplotlib) (1.2.0)
Requirement already satisfied: numpy>=1.11 in /usr/lib/python3/dist-packages
(from matplotlib) (1.17.4)
Requirement already satisfied: cycler>=0.10 in
/home/amp/.local/lib/python3.8/site-packages (from matplotlib) (0.10.0)
Requirement already satisfied: python-dateutil>=2.1 in /usr/lib/python3/dist-
packages (from matplotlib) (2.7.3)
```

1.2 1. Loading RDF

The first thing to do is to load the RDF graph we want to perform the network analysis on. By executing the next cell, we'll be asked to fill in the path to an RDF graph. This can be any path, local or online, that we can look up.

Any of the Turtle (`ttl.`) files that we include with this notebook will do; for example, `bsbm-sample.ttl`. But any Web location that leads to an RDF file (for example, the GitHub copy of that same file at https://raw.githubusercontent.com/albertmeronyo/rdf-network-analysis/master/bsbm-sample.ttl; or any other RDF file on the Web like https://raw.githubusercontent.com/albertmeronyo/lodapi/master/ghostbusters.ttl) will work too.

[14]:
```python
# RDF graph loading
path = input("Path or URI of the RDF graph to load: ")
rg = RDFGraph()
rg.parse(path, format='turtle')
print("rdflib Graph loaded successfully with {} triples".format(len(rg)))
```

```
Path or URI of the RDF graph to load: wechanged-british.ttl
rdflib Graph loaded successfully with 155 triples
```

1.3 2. Get a subgraph out of the loaded RDF graph (optional)

This cell can be skipped altogether without affecting the rest of the notebook; but it will be useful if instead of using the whole RDF grahp of the previous step, we just want to use a subgraph that's included in it.

By executing the next cell, we'll be asked two things:

- The URI of the ''entiy" type we are interested in (e.g. `http://dbpedia.org/ontology/Band`)
- The URI of the ''relation" connecting entities we are interested in (e.g. `http://dbpedia.org/ontology/influencedBy`)

Using these two, the notebook will replace the original graph with the subgraph that's constructed by those entity types and relations only.

[ ]:
```
# Subgraph construction (optional)
entity = input("Entity type to build nodes of the subgraph with: ")
relation = input("Relation type to build edges of the subgraph with: ")

# TODO: Use entity and relation as parameters of a CONSTRUCT query
query = """
PREFIX bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
CONSTRUCT {{ ?u a {} . ?u {} ?v }} WHERE {{ ?u a {} . ?u {} ?v }}""".
 ↪format(entity, relation, entity, relation)
# print(query)
subg = rg.query(query)

rg = subg
```

1.4 3. Converting rdflib.Graph to networkx.Graph

Thanks to the great work done by the rdflib developers this step, which converts the basic graph data structure of rdflib into its equivalent in networkx, is straightforward. Just run the next cell to make our RDF dataset ready for network analysis!

[15]:
```
# Conversion of rdflib.Graph to networkx.Graph
G = rdflib_to_networkx_graph(rg)
print("networkx Graph loaded successfully with length {}".format(len(G)))
```

```
networkx Graph loaded successfully with length 174
```

1.5 4. Network analysis

At this point we can run the network analysis on our RDF graph by using the networkx algorithms. Exeucting the next cell will output a full network analysis report, with the following parts:

- General network metrics (network size, pendants, density)
- Node centrality metrics (degree, eigenvector, betwenness). For these, averages, stdevs, maximum, minimum and distribution histograms are given
- Clustering metrics (connected components, clustering)
- Overall network plot

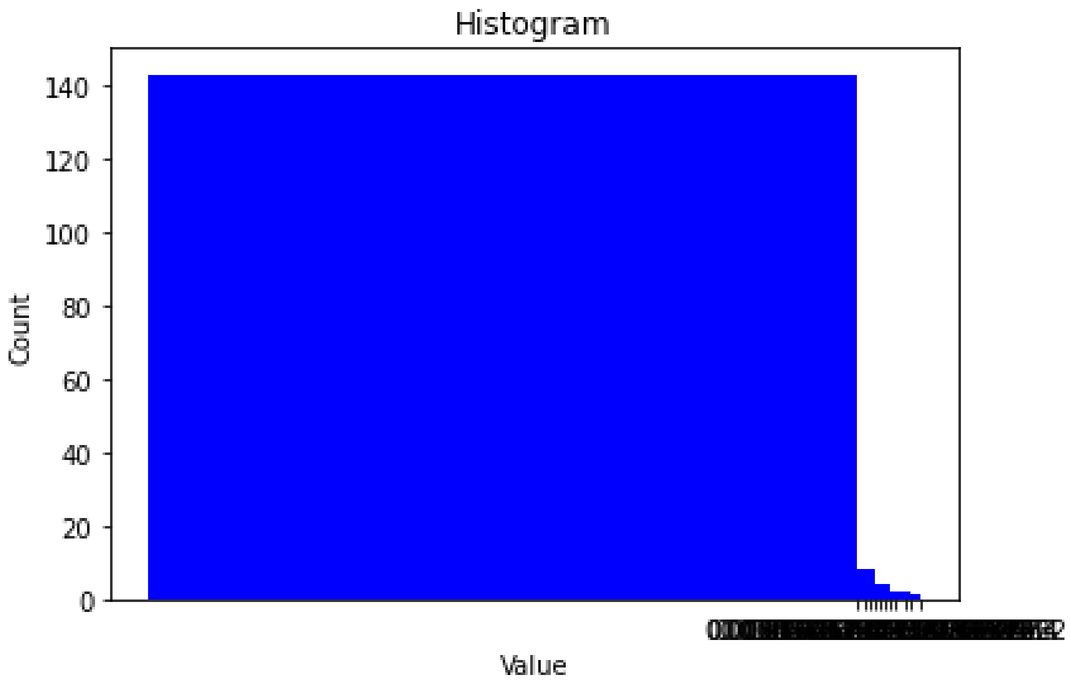
The report can be easily selected and copy-pasted for further use in other tools.

[17]:
```
# Analysis

def mean(numbers):
    return float(sum(numbers)) / max(len(numbers), 1)

def number_of_pendants(g):
    """
    Equals the number of nodes with degree 1
    """
    pendants = 0
    for u in g:
        if g.degree[u] == 1:
            pendants += 1
    return pendants


def histogram(l):
    degree_sequence = sorted([d for n, d in list(l.items())], reverse=True)
    degreeCount = collections.Counter(degree_sequence)
    deg, cnt = zip(*degreeCount.items())
    print(deg, cnt)

    fig, ax = plt.subplots()
    plt.bar(deg, cnt, width=0.80, color='b')

    plt.title("Histogram")
    plt.ylabel("Count")
    plt.xlabel("Value")
    ax.set_xticks([d + 0.4 for d in deg])
    ax.set_xticklabels(deg)

    plt.show()

# Network size
print("NETWORK SIZE")
print("============")
print("The network has {} nodes and {} edges".format(G.number_of_nodes(), G.
 ↪number_of_edges()))
print()

# Network size
print("PENDANTS")
print("============")
print("The network has {} pendants".format(number_of_pendants(G)))
print()

# Density
print("DENSITY")
print("============")
print("The network density is {}".format(nx.density(G)))
print()

# Degree centrality -- mean and stdev
dc = nx.degree_centrality(G)
degrees = []
for k,v in dc.items():
    degrees.append(v)
```

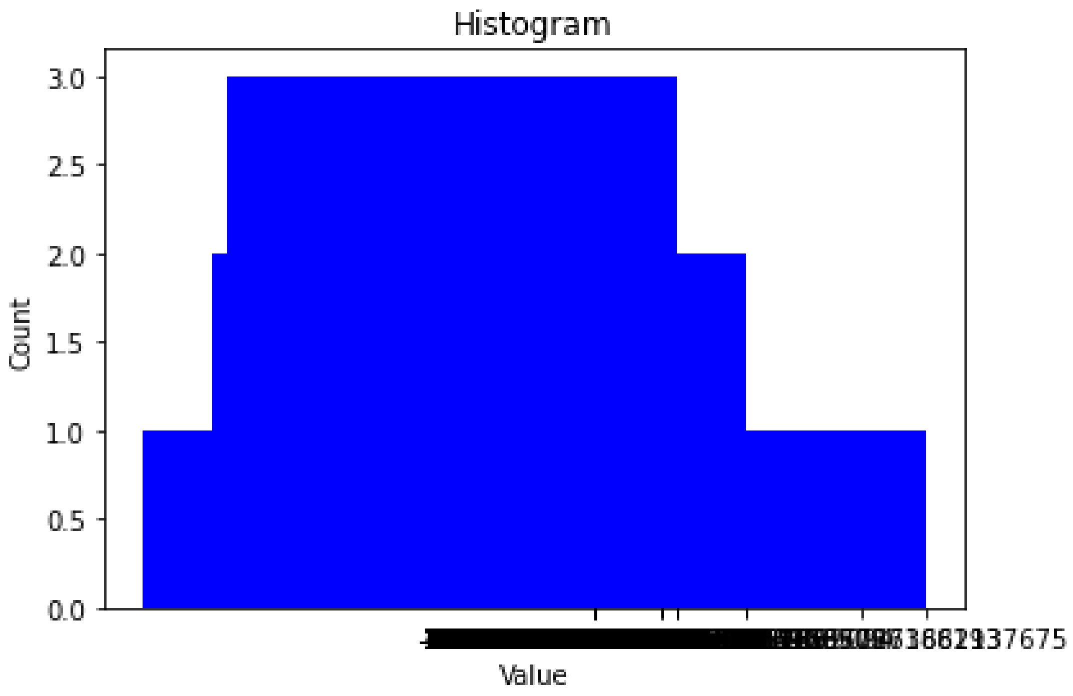
```
print("DEGREE CENTRALITY")
print("=================")
print("The mean degree centrality is {}, with stdev {}".format(mean(degrees),␣
↪statistics.stdev(degrees)))
print("The maximum node is {}, with value {}".format(max(dc, key=dc.get),␣
↪max(dc.values())))
print("The minimum node is {}, with value {}".format(min(dc, key=dc.get),␣
↪min(dc.values())))
histogram(dc)
print()

# Eigenvector centrality -- mean and stdev
ec = nx.eigenvector_centrality_numpy(G)
degrees = []
for k,v in ec.items():
    degrees.append(v)

print("EIGENVECTOR CENTRALITY")
print("======================")
print("The mean network eigenvector centrality is {}, with stdev {}".
↪format(mean(degrees), statistics.stdev(degrees)))
print("The maximum node is {}, with value {}".format(max(ec, key=ec.get),␣
↪max(ec.values())))
print("The minimum node is {}, with value {}".format(min(ec, key=ec.get),␣
↪min(ec.values())))
histogram(ec)
print()

# Betweenness centrality -- mean and stdev
# bc = nx.betweenness_centrality(G)
# degrees = []
# for k,v in bc.items():
#     degrees.append(v)
# print("BETWEENNESS CENTRALITY")
# print("======================")
# print("The mean betwenness centrality is {}, with stdev {}".
↪format(mean(degrees), statistics.stdev(degrees)))
# print("The maximum node is {}, with value {}".format(max(bc, key=bc.get),␣
↪max(bc.values())))
# print("The minimum node is {}, with value {}".format(min(bc, key=bc.get),␣
↪min(bc.values())))
# histogram(bc)
# print()
```

```python
# Connected components
cc = list(nx.connected_components(G))
print("CONNECTED COMPONENTS")
print("====================")
print("The graph has {} connected components".format(len(cc)))
for i,c in enumerate(cc):
    print("Connected component {} has {} nodes".format(i,len(c)))
print()

# Clusters
cl = nx.clustering(G)
print("CLUSTERS")
print("========")
print("The graph has {} clusters".format(len(cl)))
for i,c in enumerate(cl):
    print("Cluster {} has {} nodes".format(i,len(c)))
print()

# Plot
print("Visualizing the graph:")
plt.plot()
plt.figure(1)
nx.draw(G, with_labels=False, font_weight='normal', node_size=60, font_size=8)
plt.figure(1,figsize=(120,120))
plt.savefig('example.png', dpi=1000)
```

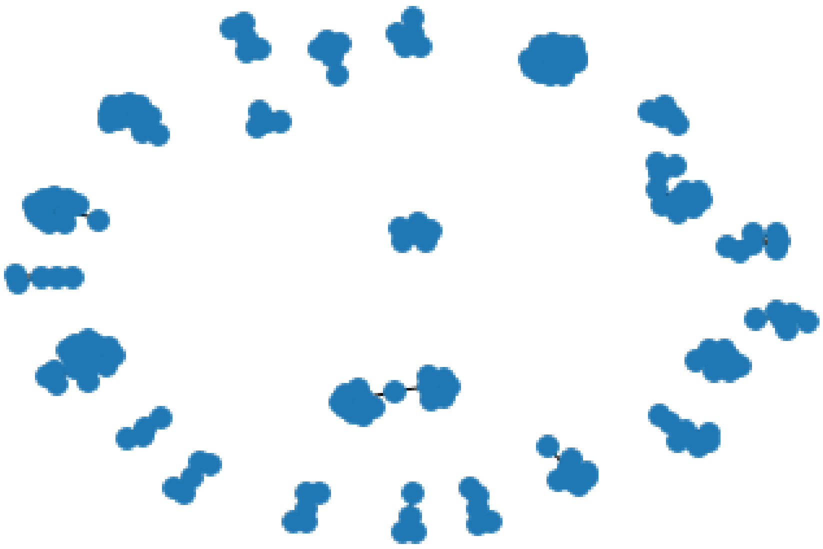
```
NETWORK SIZE
============
The network has 174 nodes and 154 edges

PENDANTS
============
The network has 143 pendants

DENSITY
============
The network density is 0.010231878280512923

DEGREE CENTRALITY
=================
The mean degree centrality is 0.010231878280512965, with stdev
0.011945260908062486
The maximum node is http://www.wikidata.org/entity/Q5373427, with value
0.07514450867052022
The minimum node is wcd_00153_id, with value 0.005780346820809248
(0.07514450867052022, 0.06358381502890173, 0.057803468208092484,
0.046242774566473986, 0.04046242774566474, 0.03468208092485549,
0.028901734104046242, 0.023121387283236993, 0.017341040462427744,
0.011560693641618497, 0.005780346820809248) (1, 2, 1, 2, 4, 2, 2, 8, 4, 5, 143)
```



```
EIGENVECTOR CENTRALITY
======================
The mean network eigenvector centrality is 0.01948514200092661, with stdev
0.07347435901965672
The maximum node is http://www.wikidata.org/entity/Q1382113, with value
0.5877661662137675
The minimum node is http://www.wikidata.org/entity/Q850141, with value
-4.505481786806643e-16
(0.5877661662137675, 0.4744595027388193, 0.26884388627369094,
0.2688438862736909, 0.1487606117642697, 0.14876061176426966,
0.14876061176426963, 0.1487606117642696, 0.12008327450942122,
0.12008327450942116, 0.12008327450942113, 1.3593413091090835e-16,
9.989522336346594e-17, 8.834850543356192e-17, 8.367196838271632e-17,
8.130062294824438e-17, 8.109920814682827e-17, 7.639250940834377e-17,
6.60755003837558e-17, 6.458331586029222e-17, 6.378391273135149e-17,
5.776875464768082e-17, 5.5230781654597724e-17, 4.632201986965634e-17,
4.5948913688461446e-17, 4.5729664583611263e-17, 4.180583512389912e-17,
3.990563383927098e-17, 3.933785144010534e-17, 3.796689971720141e-17,
3.729655473350136e-17, 3.518282345835072e-17, 3.3858485731042385e-17,
3.116169039624658e-17, 3.0462068968057656e-17, 2.96051565728719e-17,
2.928765390998258e-17, 2.8328372810121837e-17, 2.824097600371789e-17,
2.490232387743002e-17, 2.471227843456779e-17, 2.397858460927947e-17,
2.3693851342977602e-17, 2.353385305828489e-17, 1.96280254858043e-17,
1.9396502611160725e-17, 1.7312280663938313e-17, 1.706705487415864e-17,
1.677162700055526e-17, 1.6009894424159424e-17, 1.3715157481941634e-17,
1.3392917709502256e-17, 1.3282265459074696e-17, 1.2536087619363648e-17,
1.1003381032830206e-17, 1.0367419703382782e-17, 9.690003206518953e-18,
9.119550720730226e-18, 6.669266561467615e-18, 6.2847727103900965e-18,
5.925975685261885e-18, 5.6400307279347755e-18, 4.470552530857102e-18,
4.361772042560186e-18, 3.885819732618177e-18, 2.5991871601432675e-18,
2.293730052186802e-18, 2.216763629558276e-18, 1.5562080570519094e-18,
1.5101135079016543e-18, 8.735088056917535e-19, 5.091946402563726e-19,
-7.380313762569938e-20, -9.145440149184252e-20, -1.6131610381332627e-18,
-2.4637680242200984e-18, -3.6947795051584144e-18, -4.460444437411965e-18,
-4.9426428822043514e-18, -5.0168305851493625e-18, -5.9902170029824135e-18,
-6.478455626071773e-18, -7.657928674497709e-18, -7.796840606083429e-18,
-8.172173875109491e-18, -8.93151464013122e-18, -8.983016759598348e-18,
-1.0032161946479602e-17, -1.0104496019381209e-17, -1.0123396919182274e-17,
-1.1198386672833206e-17, -1.3557292598436024e-17, -1.3730489496385865e-17,
-1.4712533516235855e-17, -1.521153625256868e-17, -1.5629764138743687e-17,
-1.700655190433631e-17, -1.7044920801100344e-17, -1.7493966488533344e-17,
-1.770710757967251e-17, -1.8165666689094258e-17, -1.8892770988936693e-17,
-1.9027407083393462e-17, -2.009564437759557e-17, -2.010810736824598e-17,
-2.0648444074975188e-17, -2.082250898753491e-17, -2.141871505865804e-17,
-2.1858624313601425e-17, -2.273348040287359e-17, -2.2919382920802238e-17,
-2.4286128663675305e-17, -2.4291955939488523e-17, -2.437511230477948e-17,
-2.4619580104228264e-17, -2.5046226881248077e-17, -2.5087553268786578e-17,
-2.5717486384176555e-17, -2.7081292267934938e-17, -3.0491495862458354e-17,
-3.068317981835574e-17, -3.093754708903187e-17, -3.157642045569366e-17,
-3.2272613405298885e-17, -3.3422726711265095e-17, -3.511117111835248e-17,
-3.639011172442013e-17, -3.681622297742476e-17, -3.744625937459483e-17,
-3.7640692960354386e-17, -3.8087987972194085e-17, -3.8272236992131277e-17,
-3.881750104834077e-17, -4.0501962532597664e-17, -4.318125372162919e-17,
-4.3849139175554844e-17, -4.6004154690447276e-17, -4.8885964191348045e-17,
-4.946521968011863e-17, -5.0002431772889196e-17, -5.146258674314064e-17,
-5.800655455856678e-17, -5.925918849422934e-17, -6.154766862396167e-17,
-6.40759874005501e-17, -6.5617533442163e-17, -6.94373540330799e-17,
-7.191715408032303e-17, -7.285838599102591e-17, -7.700483730548221e-17,
-8.117098645054182e-17, -8.185599975814986e-17, -8.879302744879469e-17,
-9.083873672066412e-17, -9.446608991054129e-17, -9.852745480651999e-17,
-9.887923813067803e-17, -1.1231135809944714e-16, -1.1382903973526863e-16,
-1.162264728904461e-16, -1.2143064331837652e-16, -1.290582547627127e-16,
-1.2918140103891835e-16, -1.3250093073348863e-16, -1.446014962661383e-16,
-1.4969791370238877e-16, -1.5439038936193586e-16, -2.0566414218240156e-16,
-4.505481786806643e-16) (1, 1, 1, 2, 1, 3, 1, 2, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)
```



```
CONNECTED COMPONENTS
====================
The graph has 23 connected components
Connected component 0 has 16 nodes
Connected component 1 has 14 nodes
Connected component 2 has 5 nodes
Connected component 3 has 15 nodes
Connected component 4 has 5 nodes
Connected component 5 has 12 nodes
Connected component 6 has 11 nodes
Connected component 7 has 4 nodes
Connected component 8 has 5 nodes
Connected component 9 has 8 nodes
Connected component 10 has 4 nodes
Connected component 11 has 8 nodes
Connected component 12 has 7 nodes
Connected component 13 has 5 nodes
Connected component 14 has 11 nodes
Connected component 15 has 8 nodes
Connected component 16 has 5 nodes
Connected component 17 has 6 nodes
Connected component 18 has 5 nodes
Connected component 19 has 5 nodes
Connected component 20 has 4 nodes
Connected component 21 has 5 nodes
Connected component 22 has 6 nodes

CLUSTERS
========
The graph has 174 clusters
Cluster 0 has 39 nodes
Cluster 1 has 12 nodes
Cluster 2 has 39 nodes
Cluster 3 has 40 nodes
Cluster 4 has 12 nodes
Cluster 5 has 40 nodes
Cluster 6 has 12 nodes
Cluster 7 has 38 nodes
Cluster 8 has 39 nodes
Cluster 9 has 40 nodes
Cluster 10 has 12 nodes
Cluster 11 has 38 nodes
Cluster 12 has 25 nodes
Cluster 13 has 38 nodes
Cluster 14 has 39 nodes
Cluster 15 has 40 nodes
Cluster 16 has 40 nodes
Cluster 17 has 39 nodes
Cluster 18 has 25 nodes
Cluster 19 has 25 nodes
Cluster 20 has 38 nodes
Cluster 21 has 12 nodes
Cluster 22 has 40 nodes
Cluster 23 has 40 nodes
Cluster 24 has 40 nodes
Cluster 25 has 25 nodes
Cluster 26 has 39 nodes
Cluster 27 has 39 nodes
Cluster 28 has 12 nodes
Cluster 29 has 37 nodes
Cluster 30 has 40 nodes
Cluster 31 has 25 nodes
Cluster 32 has 39 nodes
Cluster 33 has 25 nodes
Cluster 34 has 25 nodes
Cluster 35 has 38 nodes
Cluster 36 has 12 nodes
Cluster 37 has 12 nodes
Cluster 38 has 38 nodes
Cluster 39 has 40 nodes
Cluster 40 has 40 nodes
Cluster 41 has 39 nodes
Cluster 42 has 40 nodes
Cluster 43 has 39 nodes
Cluster 44 has 12 nodes
Cluster 45 has 38 nodes
Cluster 46 has 25 nodes
Cluster 47 has 25 nodes
Cluster 48 has 25 nodes
Cluster 49 has 25 nodes
Cluster 50 has 12 nodes
Cluster 51 has 25 nodes
Cluster 52 has 40 nodes
Cluster 53 has 25 nodes
Cluster 54 has 25 nodes
Cluster 55 has 25 nodes
Cluster 56 has 38 nodes
Cluster 57 has 40 nodes
Cluster 58 has 39 nodes
Cluster 59 has 12 nodes
Cluster 60 has 39 nodes
Cluster 61 has 25 nodes
Cluster 62 has 39 nodes
Cluster 63 has 12 nodes
Cluster 64 has 39 nodes
Cluster 65 has 25 nodes
Cluster 66 has 40 nodes
Cluster 67 has 38 nodes
Cluster 68 has 12 nodes
Cluster 69 has 25 nodes
Cluster 70 has 39 nodes
Cluster 71 has 39 nodes
Cluster 72 has 40 nodes
Cluster 73 has 40 nodes
Cluster 74 has 25 nodes
Cluster 75 has 25 nodes
Cluster 76 has 40 nodes
Cluster 77 has 12 nodes
Cluster 78 has 25 nodes
Cluster 79 has 25 nodes
Cluster 80 has 39 nodes
Cluster 81 has 39 nodes
Cluster 82 has 25 nodes
Cluster 83 has 40 nodes
Cluster 84 has 25 nodes
Cluster 85 has 39 nodes
Cluster 86 has 25 nodes
Cluster 87 has 25 nodes
Cluster 88 has 39 nodes
Cluster 89 has 25 nodes
Cluster 90 has 25 nodes
Cluster 91 has 25 nodes
Cluster 92 has 39 nodes
Cluster 93 has 40 nodes
Cluster 94 has 25 nodes
Cluster 95 has 39 nodes
Cluster 96 has 39 nodes
Cluster 97 has 12 nodes
Cluster 98 has 25 nodes
Cluster 99 has 25 nodes
Cluster 100 has 25 nodes
Cluster 101 has 39 nodes
Cluster 102 has 39 nodes
Cluster 103 has 12 nodes
Cluster 104 has 25 nodes
Cluster 105 has 25 nodes
Cluster 106 has 25 nodes
Cluster 107 has 25 nodes
Cluster 108 has 38 nodes
Cluster 109 has 40 nodes
Cluster 110 has 40 nodes
Cluster 111 has 25 nodes
Cluster 112 has 39 nodes
Cluster 113 has 40 nodes
Cluster 114 has 40 nodes
Cluster 115 has 25 nodes
Cluster 116 has 12 nodes
Cluster 117 has 25 nodes
Cluster 118 has 39 nodes
Cluster 119 has 39 nodes
Cluster 120 has 40 nodes
Cluster 121 has 39 nodes
Cluster 122 has 25 nodes
Cluster 123 has 12 nodes
Cluster 124 has 40 nodes
Cluster 125 has 25 nodes
Cluster 126 has 40 nodes
Cluster 127 has 39 nodes
Cluster 128 has 40 nodes
Cluster 129 has 40 nodes
Cluster 130 has 39 nodes
Cluster 131 has 25 nodes
Cluster 132 has 25 nodes
Cluster 133 has 12 nodes
Cluster 134 has 38 nodes
Cluster 135 has 25 nodes
Cluster 136 has 39 nodes
Cluster 137 has 25 nodes
Cluster 138 has 12 nodes
Cluster 139 has 25 nodes
Cluster 140 has 25 nodes
Cluster 141 has 25 nodes
Cluster 142 has 25 nodes
Cluster 143 has 25 nodes
Cluster 144 has 12 nodes
Cluster 145 has 25 nodes
Cluster 146 has 25 nodes
Cluster 147 has 40 nodes
Cluster 148 has 39 nodes
Cluster 149 has 25 nodes
Cluster 150 has 12 nodes
Cluster 151 has 12 nodes
Cluster 152 has 25 nodes
Cluster 153 has 25 nodes
Cluster 154 has 25 nodes
Cluster 155 has 25 nodes
Cluster 156 has 40 nodes
Cluster 157 has 25 nodes
Cluster 158 has 39 nodes
Cluster 159 has 25 nodes
Cluster 160 has 39 nodes
Cluster 161 has 39 nodes
Cluster 162 has 25 nodes
Cluster 163 has 12 nodes
Cluster 164 has 25 nodes
Cluster 165 has 40 nodes
Cluster 166 has 25 nodes
Cluster 167 has 38 nodes
Cluster 168 has 12 nodes
Cluster 169 has 25 nodes
Cluster 170 has 12 nodes
Cluster 171 has 40 nodes
Cluster 172 has 40 nodes
Cluster 173 has 39 nodes

Visualizing the graph:
```



B. 19th century German speaking female editors and their kinship relations as present in Wikidata

May 15, 2020

1 Network Analysis of RDF Graphs

In this notebook we provide basic facilities for performing network analyses of RDF graphs easily with Python rdflib and networkx

We do this in 4 steps: 1. Load an arbitrary RDF graph into rdflib 2. Get a subgraph of relevance (optional) 3. Convert the rdflib Graph into an networkx Graph, as shown here 4. Get an network analysis report by running networkx's algorithms on that data structure

[13]:

1.1 0. Preparation

```
# Install required packages in the current Jupyter kernel
# Uncomment the following lines if you need to install these libraries
# If you run into permission issues, try with the --user option
import sys
# !pip install -q rdflib networkx matplotlib scipy
!{sys.executable} -m pip install rdflib networkx matplotlib scipy --user

# Imports
from rdflib import Graph as RDFGraph
from rdflib.extras.external_graph_libs import rdflib_to_networkx_graph
import networkx as nx
from networkx import Graph as NXGraph
import matplotlib.pyplot as plt
import statistics
import collections
```

```
Requirement already satisfied: rdflib in /home/amp/.local/lib/python3.8/site-
packages (5.0.0)
Requirement already satisfied: networkx in /home/amp/.local/lib/python3.8/site-
packages (2.4)
Requirement already satisfied: matplotlib in
/home/amp/.local/lib/python3.8/site-packages (3.2.1)
Requirement already satisfied: scipy in /home/amp/.local/lib/python3.8/site-
packages (1.4.1)
Requirement already satisfied: pyparsing in /usr/lib/python3/dist-packages (from
rdflib) (2.4.6)
Requirement already satisfied: six in /usr/lib/python3/dist-packages (from rdflib)
(1.14.0)
```

```
Requirement already satisfied: isodate in /home/amp/.local/lib/python3.8/site-
packages (from rdflib) (0.6.0)
Requirement already satisfied: decorator>=4.3.0 in /usr/lib/python3/dist-
packages (from networkx) (4.4.2)
Requirement already satisfied: kiwisolver>=1.0.1 in
/home/amp/.local/lib/python3.8/site-packages (from matplotlib) (1.2.0)
Requirement already satisfied: numpy>=1.11 in /usr/lib/python3/dist-packages
(from matplotlib) (1.17.4)
Requirement already satisfied: cycler>=0.10 in
/home/amp/.local/lib/python3.8/site-packages (from matplotlib) (0.10.0)
Requirement already satisfied: python-dateutil>=2.1 in /usr/lib/python3/dist-
packages (from matplotlib) (2.7.3)
```

1.2 1. Loading RDF

The first thing to do is to load the RDF graph we want to perform the network analysis on. By executing the next cell, we'll be asked to fill in the path to an RDF graph. This can be any path, local or online, that we can look up.

Any of the Turtle (`ttl.`) files that we include with this notebook will do; for example, `bsbm-sample.ttl`. But any Web location that leads to an RDF file (for example, the GitHub copy of that same file at https://raw.githubusercontent.com/albertmeronyo/rdf-network-analysis/master/bsbm-sample.ttl; or any other RDF file on the Web like https://raw.githubusercontent.com/albertmeronyo/lodapi/master/ghostbusters.ttl) will work too.

[18]:
```
# RDF graph loading
path = input("Path or URI of the RDF graph to load: ")
rg = RDFGraph()
rg.parse(path, format='turtle')
print("rdflib Graph loaded successfully with {} triples".format(len(rg)))
```

```
Path or URI of the RDF graph to load: wechanged-german.ttl
rdflib Graph loaded successfully with 53 triples
```

1.3 2. Get a subgraph out of the loaded RDF graph (optional)

This cell can be skipped altogether without affecting the rest of the notebook; but it will be useful if instead of using the whole RDF grahp of the previous step, we just want to use a subgraph that's included in it.

By executing the next cell, we'll be asked two things:

- The URI of the ''entiy'' type we are interested in (e.g. `http://dbpedia.org/ontology/Band`)
- The URI of the ''relation'' connecting entities we are interested in (e.g. `http://dbpedia.org/ontology/influencedBy`)

Using these two, the notebook will replace the original graph with the subgraph that's constructed by those entity types and relations only.

```
[ ]: # Subgraph construction (optional)
entity = input("Entity type to build nodes of the subgraph with: ")
relation = input("Relation type to build edges of the subgraph with: ")

# TODO: Use entity and relation as parameters of a CONSTRUCT query
query = """
PREFIX bsbm: <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/>
CONSTRUCT {{ ?u a {} . ?u {} ?v }} WHERE {{ ?u a {} . ?u {} ?v }}""".
 ↪format(entity, relation, entity, relation)
# print(query)
subg = rg.query(query)

rg = subg
```

1.4 3. Converting rdflib.Graph to networkx.Graph

Thanks to the great work done by the rdflib developers this step, which converts the basic graph data structure of rdflib into its equivalent in networkx, is straightforward. Just run the next cell to make our RDF dataset ready for network analysis!

```
[19]: # Conversion of rdflib.Graph to networkx.Graph
G = rdflib_to_networkx_graph(rg)
print("networkx Graph loaded successfully with length {}".format(len(G)))
```

```
networkx Graph loaded successfully with length 65
```

1.5 4. Network analysis

At this point we can run the network analysis on our RDF graph by using the networkx algorithms. Exeucting the next cell will output a full network analysis report, with the following parts:

- General network metrics (network size, pendants, density)
- Node centrality metrics (degree, eigenvector, betwenness). For these, averages, stdevs, maximum, minimum and distribution histograms are given
- Clustering metrics (connected components, clustering)
- Overall network plot

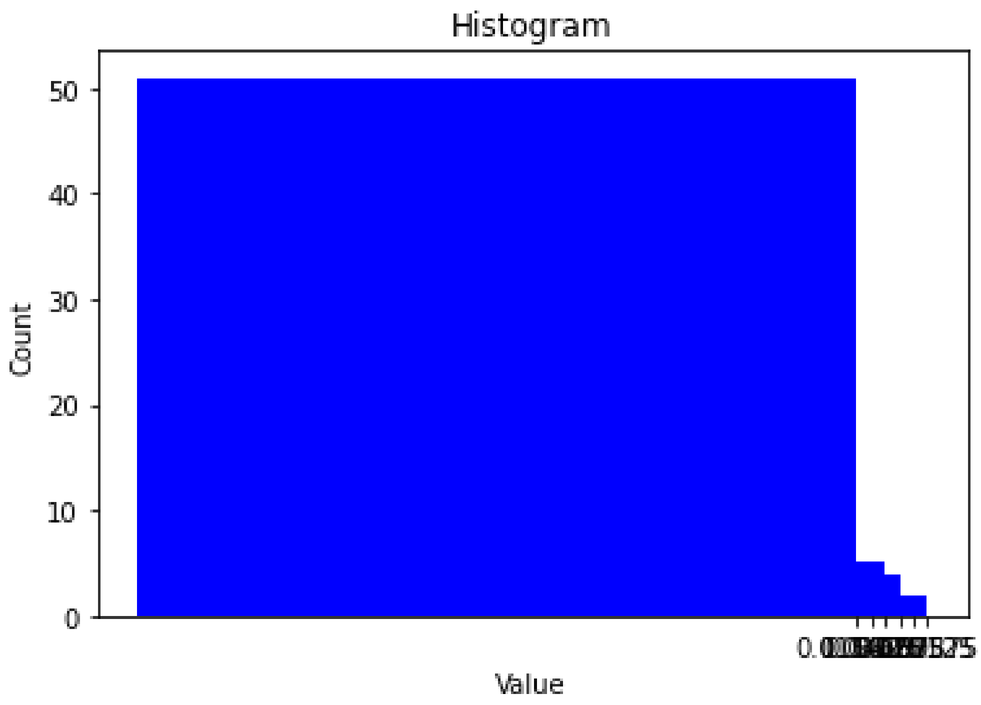
The report can be easily selected and copy-pasted for further use in other tools.

```
[20]: # Analysis

def mean(numbers):
    return float(sum(numbers)) / max(len(numbers), 1)

def number_of_pendants(g):
    """
    Equals the number of nodes with degree 1
    """
    pendants = 0
    for u in g:
        if g.degree[u] == 1:
            pendants += 1
    return pendants


def histogram(l):
    degree_sequence = sorted([d for n, d in list(l.items())], reverse=True)
    degreeCount = collections.Counter(degree_sequence)
    deg, cnt = zip(*degreeCount.items())
    print(deg, cnt)

    fig, ax = plt.subplots()
    plt.bar(deg, cnt, width=0.80, color='b')

    plt.title("Histogram")
    plt.ylabel("Count")
    plt.xlabel("Value")
    ax.set_xticks([d + 0.4 for d in deg])
    ax.set_xticklabels(deg)

    plt.show()

# Network size
print("NETWORK SIZE")
print("============")
print("The network has {} nodes and {} edges".format(G.number_of_nodes(), G.
 ↪number_of_edges()))
print()

# Network size
print("PENDANTS")
print("============")
print("The network has {} pendants".format(number_of_pendants(G)))
print()

# Density
print("DENSITY")
print("============")
print("The network density is {}".format(nx.density(G)))
print()

# Degree centrality -- mean and stdev
dc = nx.degree_centrality(G)
degrees = []
for k,v in dc.items():
    degrees.append(v)
```

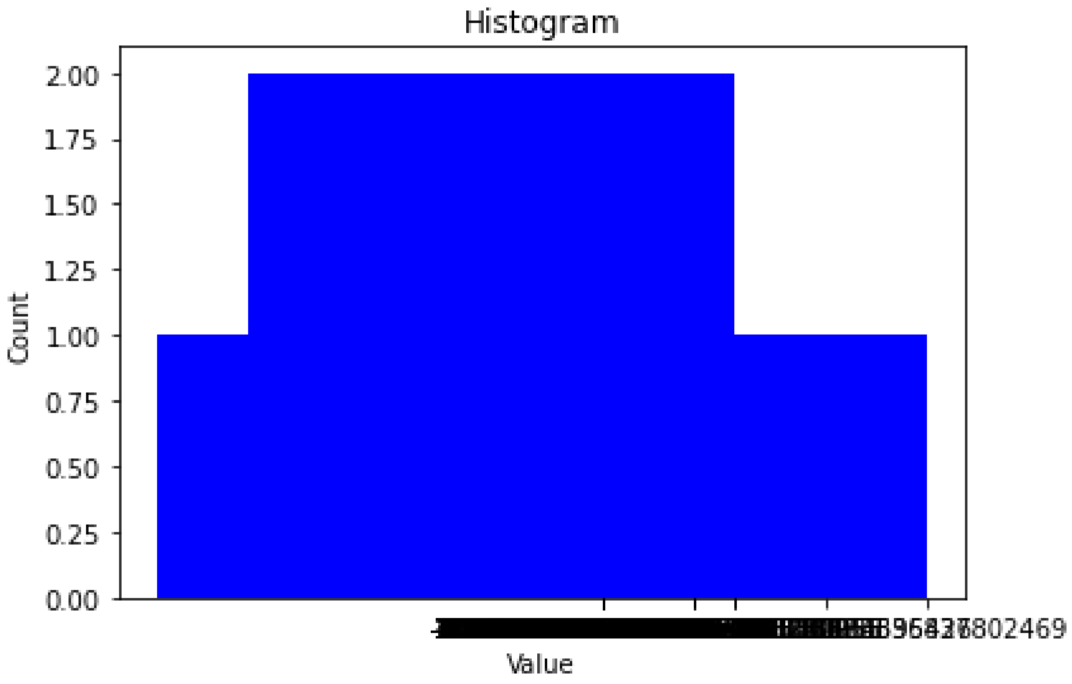
```
print("DEGREE CENTRALITY")
print("=================")
print("The mean degree centrality is {}, with stdev {}".format(mean(degrees),␣
↪statistics.stdev(degrees)))
print("The maximum node is {}, with value {}".format(max(dc, key=dc.get),␣
↪max(dc.values())))
print("The minimum node is {}, with value {}".format(min(dc, key=dc.get),␣
↪min(dc.values())))
histogram(dc)
print()

# Eigenvector centrality -- mean and stdev
ec = nx.eigenvector_centrality_numpy(G)
degrees = []
for k,v in ec.items():
    degrees.append(v)

print("EIGENVECTOR CENTRALITY")
print("======================")
print("The mean network eigenvector centrality is {}, with stdev {}".
↪format(mean(degrees), statistics.stdev(degrees)))
print("The maximum node is {}, with value {}".format(max(ec, key=ec.get),␣
↪max(ec.values())))
print("The minimum node is {}, with value {}".format(min(ec, key=ec.get),␣
↪min(ec.values())))
histogram(ec)
print()

# Betweenness centrality -- mean and stdev
# bc = nx.betweenness_centrality(G)
# degrees = []
# for k,v in bc.items():
#     degrees.append(v)
# print("BETWEENNESS CENTRALITY")
# print("======================")
# print("The mean betwenness centrality is {}, with stdev {}".
↪format(mean(degrees), statistics.stdev(degrees)))
# print("The maximum node is {}, with value {}".format(max(bc, key=bc.get),␣
↪max(bc.values())))
# print("The minimum node is {}, with value {}".format(min(bc, key=bc.get),␣
↪min(bc.values())))
# histogram(bc)
# print()
```

```python
# Connected components
cc = list(nx.connected_components(G))
print("CONNECTED COMPONENTS")
print("====================")
print("The graph has {} connected components".format(len(cc)))
for i,c in enumerate(cc):
    print("Connected component {} has {} nodes".format(i,len(c)))
print()

# Clusters
cl = nx.clustering(G)
print("CLUSTERS")
print("========")
print("The graph has {} clusters".format(len(cl)))
for i,c in enumerate(cl):
    print("Cluster {} has {} nodes".format(i,len(c)))
print()

# Plot
print("Visualizing the graph:")
plt.plot()
plt.figure(1)
nx.draw(G, with_labels=False, font_weight='normal', node_size=60, font_size=8)
plt.figure(1,figsize=(120,120))
plt.savefig('example.png', dpi=1000)
```

```
NETWORK SIZE
============
The network has 65 nodes and 53 edges

PENDANTS
============
The network has 51 pendants

DENSITY
============
The network density is 0.02548076923076923

DEGREE CENTRALITY
=================
The mean degree centrality is 0.02548076923076923, with stdev
0.020774719730186218
The maximum node is http://www.wikidata.org/entity/Q165824, with value 0.09375
The minimum node is wcd_00814_id, with value 0.015625
(0.09375, 0.078125, 0.0625, 0.046875, 0.03125, 0.015625) (2, 2, 4, 5, 1, 51)
```



```
EIGENVECTOR CENTRALITY
======================
The mean network eigenvector centrality is 0.052190328754421186, with stdev
0.11339581003839311
The maximum node is http://www.wikidata.org/entity/Q165824, with value
0.5823336837802469
The minimum node is http://www.wikidata.org/entity/Q2653682, with value
-4.221865487293616e-16
(0.5823336837802469, 0.40110781684595426, 0.23773673088280958,
0.23773673088280955, 0.23773673088280953, 0.2377367308828095,
0.16375158051905314, 0.1637515805190531, 0.16375158051905309,
0.16375158051905306, 0.16375158051905303, 4.0375682011403296e-16,
3.867620361637153e-16, 2.423140802377901e-16, 2.1493997183573874e-16,
2.0826656295842276e-16, 1.8925354328681477e-16, 1.860937486981313e-16,
1.7617115305582745e-16, 1.667799235809163e-16, 1.5837406027033936e-16,
1.3001507409184495e-16, 1.2555668835368535e-16, 1.1354908529513395e-16,
1.0195879145351594e-16, 9.659051261599858e-17, 8.249547678468506e-17,
7.150789936020287e-17, 6.477197106861316e-17, 6.34748456895395e-17,
6.255159781101699e-17, 5.748386506242491e-17, 4.5070823959909377e-17,
4.028075437461217e-17, 3.191177899331292e-17, 1.7134628208983114e-17,
1.504830421598233e-17, 1.5222146334878828e-18, -1.2422309969230075e-18,
-8.56840964600123e-18, -1.0566991786028656e-17, -1.3380020371347866e-17,
-1.82290962330364e-17, -2.5459354322188953e-17, -2.9381498680853597e-17,
-5.607146449104495e-17, -6.208476377865393e-17, -6.278772145308986e-17,
-1.0752076302444307e-16, -1.0828082789917711e-16, -1.7349787989201016e-16,
-1.765445338168193e-16, -1.879654759105251e-16, -1.9130239507871805e-16,
-1.93439031608548e-16, -1.9861678449786301e-16, -1.9935548922841931e-16,
-2.3141199604139336e-16, -2.477318548940688e-16, -2.722433427921724e-16,
-3.7706856978891896e-16, -4.221865487293616e-16) (1, 1, 1, 1, 2, 2, 1, 1, 1, 2,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)
```



```
CONNECTED COMPONENTS
====================
The graph has 12 connected components
Connected component 0 has 5 nodes
Connected component 1 has 7 nodes
Connected component 2 has 5 nodes
Connected component 3 has 4 nodes
Connected component 4 has 9 nodes
Connected component 5 has 4 nodes
Connected component 6 has 6 nodes
Connected component 7 has 6 nodes
Connected component 8 has 4 nodes
Connected component 9 has 7 nodes
Connected component 10 has 4 nodes
Connected component 11 has 4 nodes
```

```
CLUSTERS
========
The graph has 65 clusters
Cluster 0 has 37 nodes
Cluster 1 has 12 nodes
Cluster 2 has 38 nodes
Cluster 3 has 37 nodes
Cluster 4 has 37 nodes
Cluster 5 has 25 nodes
Cluster 6 has 37 nodes
Cluster 7 has 25 nodes
Cluster 8 has 39 nodes
Cluster 9 has 40 nodes
Cluster 10 has 39 nodes
Cluster 11 has 25 nodes
Cluster 12 has 25 nodes
Cluster 13 has 37 nodes
Cluster 14 has 25 nodes
Cluster 15 has 25 nodes
Cluster 16 has 39 nodes
Cluster 17 has 25 nodes
Cluster 18 has 25 nodes
Cluster 19 has 39 nodes
Cluster 20 has 12 nodes
Cluster 21 has 37 nodes
Cluster 22 has 12 nodes
Cluster 23 has 12 nodes
Cluster 24 has 25 nodes
Cluster 25 has 37 nodes
Cluster 26 has 12 nodes
Cluster 27 has 25 nodes
Cluster 28 has 25 nodes
Cluster 29 has 37 nodes
Cluster 30 has 12 nodes
Cluster 31 has 25 nodes
Cluster 32 has 39 nodes
Cluster 33 has 12 nodes
Cluster 34 has 25 nodes
Cluster 35 has 40 nodes
Cluster 36 has 25 nodes
Cluster 37 has 12 nodes
Cluster 38 has 40 nodes
Cluster 39 has 12 nodes
Cluster 40 has 25 nodes
Cluster 41 has 39 nodes
Cluster 42 has 25 nodes
Cluster 43 has 38 nodes
Cluster 44 has 12 nodes
Cluster 45 has 25 nodes
Cluster 46 has 25 nodes
Cluster 47 has 25 nodes
Cluster 48 has 25 nodes
Cluster 49 has 25 nodes
Cluster 50 has 25 nodes
Cluster 51 has 37 nodes
Cluster 52 has 25 nodes
Cluster 53 has 12 nodes
Cluster 54 has 25 nodes
Cluster 55 has 40 nodes
Cluster 56 has 40 nodes
Cluster 57 has 40 nodes
Cluster 58 has 25 nodes
Cluster 59 has 12 nodes
Cluster 60 has 38 nodes
Cluster 61 has 25 nodes
Cluster 62 has 12 nodes
Cluster 63 has 25 nodes
Cluster 64 has 25 nodes

Visualizing the graph:
```