But How Do We Store It? (Big) Data Architecture in the Social-Scientific Research Process

Günther, E.; Trilling, D.; van de Velde, B.

Publication date
2018

Document Version
Submitted manuscript

Published in
Computational Social Science in the Age of Big Data

Link to publication

Citation for published version (APA):
Günther, E., Trilling, D., & van de Velde, B. (2018). But How Do We Store It? (Big) Data Architecture in the Social-Scientific Research Process. In C. M. Stuetzer, M. Welker, & M. Egger (Eds.), *Computational Social Science in the Age of Big Data: Concepts, Methodologies, Tools, and Applications* (pp. 161-187). (Neue Schriften Zur Online-Forschung; Vol. 15). Herbert von Halem Verlag.

But how do we store it? (Big) data architecture in the social-scientific research process

Damian Trilling
d.c.trilling@uva.nl, +31-648133576
Department of Communication Science, University of Amsterdam
Postbus 15791, 1001 NG Amsterdam, Netherlands

Bob van de Velde
R.N.vandeVelde@uva.nl, +31-653396804
Informatics Institute, University of Amsterdam
Postbus 94323, 1090 GH Amsterdam, Netherlands

Elisabeth Günther
elisabeth.guenther@uni-muenster.de, +49-2518329236
Department of Communication, University of Münster
Bispinghof 9-14, 48143 Münster, Germany

Abstract. *The social-scientific research process is usually considered to consist of reviewing literature and theory, followed by the generation of research questions or hypotheses, the collection of data, their analysis, and writing up the findings. In this chapter, we argue that in the age of Big Data, social scientists have to increasingly consider a step that can be located between the collection and analysis of data: the storage of the data. Based on the notion of data architecture, we discuss how the choices made at this stage impact the ways the data can be used and the research questions that can be answered. In particular, we compare file dumps, relational databases, document stores, and graph databases. We develop a scheme to make a choice for one of these approaches based on four criteria: the need for preprocessing, the properties of the data, the research design, the available infrastructure, and the available expertise. We conclude by summarizing their strengths and weaknesses along two dimensions: ease-of-storage versus reliability-of-retrieval and ease-of-use versus power-to-explore.*

Introduction

Throughout the last decades, a large body of social-scientific methodological literature on both data *collection* and data *analysis* has been accumulated. With the emergence of computational methods in the social sciences, both parts have been addressed and new methods and techniques have been incorporated into the methodological toolbox of relevant disciplines. For instance, there is literature on the different ways to *collect* (e.g., Trilling, 2014), *clean* (e.g., Günther & Scharkow, 2014) and *analyze* large-scale data sets (e.g., Boumans & Trilling, 2016; Burscher, Odijk, Vliegenthart, de Rijke, & de Vreese, 2014;

Grimmer & Stewart, 2013; Günther & Quandt, 2016; Jacobi, van Atteveldt, & Welbers, 2016; Maireder, Ausserhofer, Schumann, & Taddicken, 2015). However, there is a lack of attention to the step in between: the management and the architecture of the data. While this step did not need much attention in the "old" paradigm of manual analysis (storing printed newspaper copies is a trivial task), making the right choices in how to organize and store the data has important implication in the "new" paradigm of computational analysis of large-scale datasets (see also Kitchin, 2014).

In this chapter, we argue that for the computational social scientist – next to technical skills for data collection and data analysis, such as data mining or machine learning techniques –, being able to make informed decisions regarding the data architecture is equally important: As a prerequisite to conduct analyses of these types, researchers need to be able to organize and manage the underlying data. When working with large and complex datasets, *a priori* choices have big implications and are hard to undo. Problems can include lost data, overwhelming analysis times, and misleading analyses. We first introduce a model that illustrates how considerations related to data architecture fit into the social-scientific research process. We then discuss different criteria that have to be considered and how they relate to different data models and paradigms.

A prototypical use case might help to illustrate the consequences of different choices that can be made. Let us assume we want to conduct a large-scale analysis of (online) news items. Only one or two years of coverage from the major news outlets of one single country easily amounts to hundreds of thousands of news items, and it is not uncommon to have millions of articles. But although the mere size of such a dataset requires new ways of thinking about its storage, its structure is not fundamentally different from traditional approaches. This changes when we combine data from different sources, when items are nested in each other, or when different types of items are part of the dataset.

When dealing with online news articles, for instance, simple data models and storage solutions become problematic when we want to add user comments to the data set. This addition makes the data structure considerably more complicated, as it is no longer possible to model our data in a tabular format with columns that are the same for every entry. Additionally, comments are nested within the articles (each entry features additional comments in reply), meaning that we also need to consider the hierarchical relations between the comments. Social media data often have a similar structure: They also consist of posts written by users, and replies by others to these posts. If one does not want to loose this information, one has to think of how to store these relations.

All of this is not an unusual setup for a social scientific research project, but managing data of this size and structure certainly is – information with varying features and complex *relations* need to be incorporated in a reliable infrastructure adequate for Big Data analyses: *So, how do we store it?*

In this chapter, we take up the challenge to find a balance between providing hands-on guidance for the social scientist on the one hand, and going into some necessary technical details on the other hand. We decided to keep the most important technical terms in order to enable the interested reader to look for further literature, and additionally provide a table where these terms are defined and where examples are given.

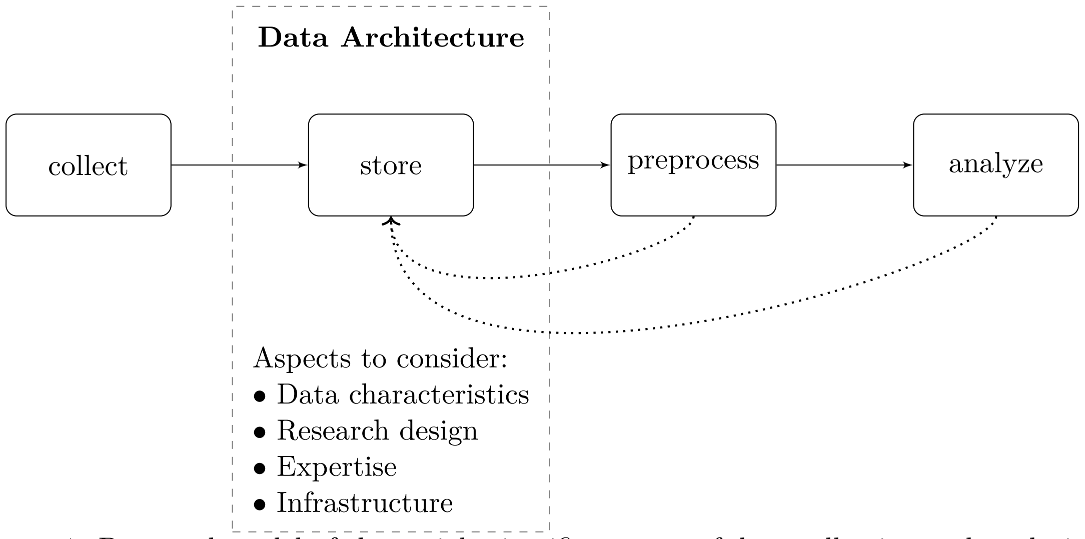


Figure 1. Proposed model of the social-scientific process of data collection and analysis in the age of Big Data. As the dotted arrows indicate, the results of preprocessing and analyis are preferably fed back into the storage system for further re-use.

Data Architecture

Traditionally, data storage and data analysis are not explicitly addressed as elements of the social-scientific research process. For instance, Bryman (2016) lists "literature review; formulating concepts and theories; devising research questions; sampling; data collection; data analysis; and writing up findings" (p. 2) as elements of the research process. Similarly, Field (2016) presents a flow-chart, according to which the generation of a research question is followed by a consulation of theory, the generation of hypotheses and predictions, the collection of data, their analysis, and the generalization of the results (p. 15). We argue that in the era of Big Data, these models have to be extended to include "data architecture" as an additional and independent step located between "data collection" and "data analysis" (see Fig. 1). Often, Big Data are described by *the four Vs* – high volume (i.e., they occupy a lot of storage), high velocity (i.e., they change quickly), high variety (i.e., they may have a heterogeneous structure), and unclear veracity (i.e., they may contain errors). Although in many cases of big datasets, not all four Vs are present (Kitchin & McArdle, 2016), these characteristics are directly relevant for a social scientist designing a data architecture: The *volume* of data may require scalable storage, especially as the *velocity* of data additions and data mutations increases. The *variety* of data can require flexible storage solutions or complex relational models to leverage the interconnectedness of much new data. The *veracity* may require additional provenance information above and beyond classical metadata.

In this chapter, we provide guidelines to allow researchers without a background in IT to make informed decisions on how to deal with such data. A simplified model of the social-scientific research process in the age of Big Data is illustrated in Figure 1: Data that are collected have to be stored in some way before they can be preprocessed and analyzed. In addition, as the dotted lines in Figure 1 indicate, it is often also desirable that the results of the preprocessing and/or analysis steps can be fed back into the database for future use.

By referring to these tasks in the social-scientific research process as problems of "data architecture", we follow the definiton by Comella-Dorda, Lewis, Place, Plakosh, and Searcord (2001), who write: "Data architecture defines how data is stored, managed, and used in a system" (p. 1). More specifically, according to Inmon and Linstedt (2014), the "four most interesting aspects of data architecture are: 1. The pysical manifestation of data; 2. The logical linkage of data; 3. The internal format of data; 4. The file structure of data." (p. 199). To illustrate the importance of data architecture, Tupper (2011) compares it with the role of architecture in the real world: learning how to create sustainable buildings was a precondition for mankind to evolve. And, one might argue, a well-designed data architecture is the precondition for insightful analysis and scientific research to evolve.

Not all of these aspects are equally important for a social scientist; and some of them are mainly interesting from a technical standpoint. In the following section, we will zoom in on those aspects in which social scientists typically have to make an informed choice in order to get out most for their analyses.

Considerations and choices

We propose to distinguish five main factors guiding the design of the data architecture for any given project (see Fig. 2): When designing a data architecure, social scientists have to reflect on a number of questions regarding the characteristics of the data (I & II), the research design (III), the available expertise (IV), and finally, the existing and/or available infrastructure (V). The answers to these question cannot be given independently from each other. For instance, if one of the goals of the project is the construction of a long-term database to be used by multiple projects, the requirements regarding the cleaning and preprocessing of the data will differ from a one-shot project (feedback loop between data and research design); the research design is also interlinked with the project infrastructure, most importantly since it outlines the requirements of the analysis operations carried out in later stages of the project (feedback loop between research design and infrastructure). In the design of the data architecture, project-specific interdependencies like these are equally important to decide how the data in the storage should look like.

Different databases come with their own strengths and weaknesses, which have major implications for the later steps in the research process. Strengths could be be ease-of-storage, ease-of-retrieval, learning-curve and ability-to-scale. Weaknesses could be related to the flexibility, reliability, effectiveness and compatibility of a system. In order to match a research project with an adequate system, answering the questions described in Figure 2 helps us to a) define our project-specific needs and b) evaluate the relative importance of the strengths and weaknesses of each system. To establish a basic knowledge on the available options and their modes of operation, throughout this chapter, we focus on four different types of systems:

1. File dumps

2. Relational Databases such as MySQL,

3. Document stores such as MongoDB,

4. Graph databases such as Neo4j.

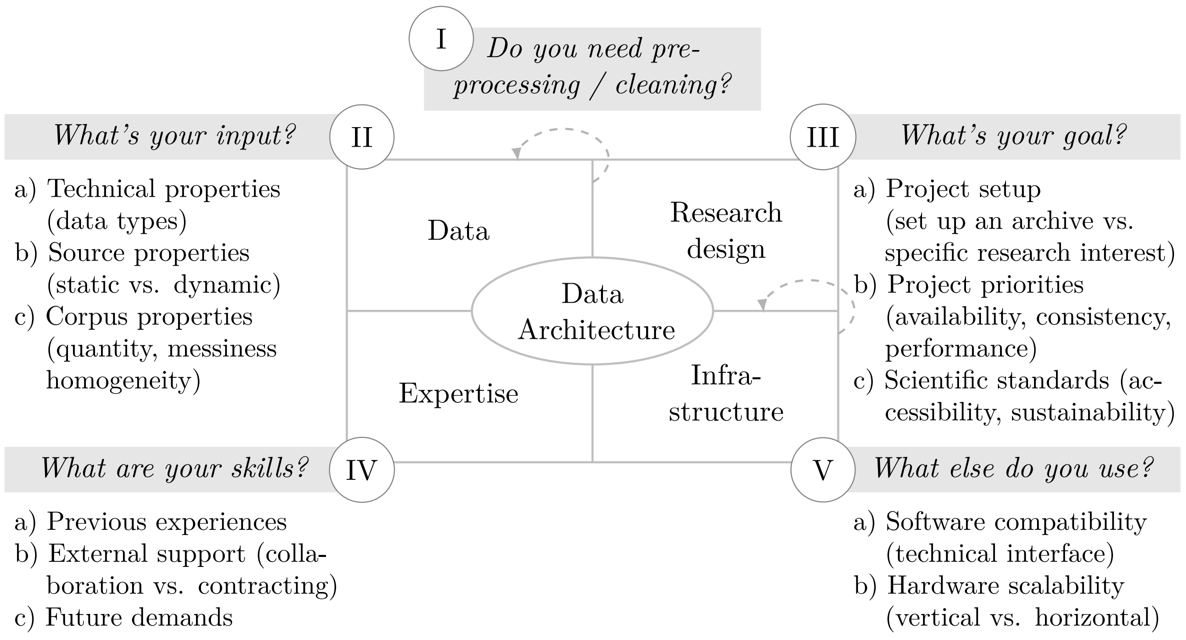

I
Do you need pre-processing / cleaning?
II
What's your input?
a) Technical properties (data types)
b) Source properties (static vs. dynamic)
c) Corpus properties (quantity, messiness homogeneity)
III
What's your goal?
a) Project setup (set up an archive vs. specific research interest)
b) Project priorities (availability, consistency, performance)
c) Scientific standards (accessibility, sustainability)
Data
Research design
Data Architecture
Expertise
Infra-structure
IV
What are your skills?
a) Previous experiences
b) External support (collaboration vs. contracting)
c) Future demands
V
What else do you use?
a) Software compatibility (technical interface)
b) Hardware scalability (vertical vs. horizontal)

Figure 2. Central factors influencing the design of the data architecture.

File dumps are the most simple approach of storing data: in this approach, each unit of observation (each article, page of comments, . . .) is simply saved to a file. While they can be cumbersome and inefficient to analyze, file dumps can be a useful additional fallback option due to their simplicity. *Relational Databases* are databases that contain a set of interlinked tables. Each table has a fixed number of columns, and each row as a so-called key, which allows researchers to store and connect information with different features from different tables. *Document stores* refer to databases that do not rely on such tables, which is why they are also referred to as NoSQL databases. In contrast, each entry is seen as a document, which has a set of keys with associated values. For instance, a key could be 'title' or 'published_at', and their values could be 'LinkedIn to get Banned in Russia' and 'October 26, 2016'. In contrast to relational databases with their fixed column structure, the keys do not necessarily have to be the same for each document in a document sore. *Graph databases* are an extension of the previous type. Next to nodes, which are in fact sets of key-value pairs just like in a document store, graph databases also store edges, i.e. connections between the nodes. For example, node 1 might represent a specific article, and node 2 might represent a person, with the link between the two indicating that the person is mentioned in the article.

I. Prelude – *Do you need preprocessing and cleaning?*

The considerations and choices to be made regarding the data architecture typically start with the question whether the data need to be processed in any way before being stored (see Fig. 2, I). Often, a lot more data has been collected than what is actually needed when crawled from online sources, meaning that relevant pieces of information are mixed with irrelevant content. One could argue that this is less of a problem nowadays, because recent technological advances have made it possible to store and query enormous quantities of unstructured data in an efficient manner. For example, tools like ElasticSearch make it possible to use free-form text search of huge collections of texts. But while there is a general trend towards such unstructured data stores, the underlying philosophy can be at odds with the traditional social-scientific research process. In social-scientific research, one typically wants to know *why* a certain outcome occurs. Therefore, rather than constructing models with an extremely high number of features, one would typically focus on a smaller number of theoretically interesting features, even if this goes at the expense of maximizing the predictive value of the model. This means that in social-scientific research, there is an inherent need for having *structured* data sets. Therefore, social scientists usually have clear ideas, guided by theory, about which features need to be extracted from a dataset.

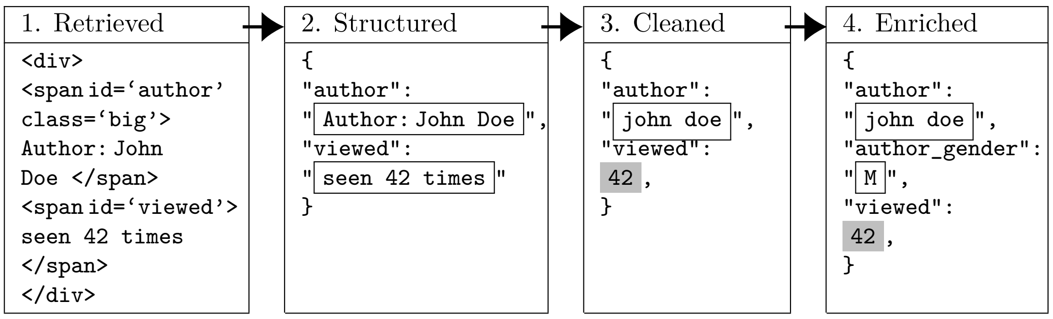
Considerations regarding data cleaning first and foremost depend on the objective of the research project (see Fig. 2, III): If the goal is to set up an archive as a basis for various, maybe not yet known analyses, it might be preferable to store as much information as possible and take care about extracting relevant information in later steps of the research process. If the research interest is more specific, unneeded data can be discarded early on to allow for a more efficient data structure.

In case data cleaning is included in the process, there are two ways to look at it: One is to *extract relevant* parts of the news website, the other to *remove irrelevant* information. Typically, we need a combination of both, starting with the noise caused by the HTML page structure and advertisements. This so-called boilerplate content makes up a major part of most web pages and is irrelevant to most social-scientific research projects.

There are several strategies that can be applied (e.g., Günther & Scharkow, 2014; Pomikálek, 2011): First, relevant content – in our example, the article text, reader comments, and metadata – can be targeted based on HTML-tags, XPATHs, and CSS selectors that guide the appearance of a website. Exploiting this structure, website-specific regular expressions and/or libraries such as Python's *Beautiful Soup* (Richardson, 2015) and *lxml* (Behnel, Faassen, & Bicking, 2016) can be used to extract relevant content. Another strategy is to make use of available print versions, which are often supplied by a website as a service to its users. Lastly, there are several algorithmic solutions that, e. g. in the case of the *Boilerpipe* library, make use of the ratio of hyperlink- to non-hyperlink-words in a paragraph (for more information, see Kohlschütter, Fankhauser, & Nejdl, 2010).

Preprocessing steps can refer to the stemming of words, the extraction of hyperlinks, or the detection of named entities that are mentioned in the news articles. When working with large datasets, deciding which preprocessing steps to take during the data storage is an important consideration: While this slows down the process during these initial stages, the performance of further analyses will benefit if they can build upon a preprocessed version of the collected data. In other cases, preprocessing might even be essential to enable further steps of the data collection. External sources such as Wikipedia can be used to augment information on people or organizations that were detected as named entities. Another research interest might target extracted hyperlinks to expand the sample in a snowball approach (e.g. Waldherr, Maier, Miltner, & Günther, 2016).

Figure 3 illustrates a possible preprocessing workflow. When analyzing web data such as online news and comments, they are often retrieved as HTML pages (1.). This data is



```
<div>
<span id='author'
class='big'>
Author: John
Doe </span>
<span id='viewed'>
seen 42 times
</span>
</div>
```

```
{
"author":
"Author: John Doe",
"viewed":
"seen 42 times"
}
```

```
{
"author":
"john doe",
"viewed":
42,
}
```

```
{
"author":
"john doe",
"author_gender":
"M",
"viewed":
42,
}
```

Figure 3. Data processing flow. Data types are indicated by a frame for strings of text and a shade for integer numbers.

then structured by mapping keys to values from the HTML (2.). Data cleaning entails getting the data types right and removing noise such as uninformative capitalization and converting strings that contain numbers to numbers (3.). Data can then be enriched by adding properties that can be deduced, but are not included in, original fields, such as the gender of a name by looking at external resources that map names to genders (4.). Depending on the goals of the project (Fig. 2, III), data storage can happen at any stage in this process.

II. Data – *What's your input?*

There is no one right way to store large datasets – the data architecture always needs to be tailored to the specific data at hand. There are, however, some recurring considerations that guide the process towards finding the right data structure for a project. As described in Figure 2, this depends on various properties of the data, starting on a technical level with the most basic choices regarding data types (see Fig. 2, IIa).

Data types. While there is no consensus on what exactly defines Big Data (see, e.g., Kitchin & McArdle, 2016), one characteristic of many such datasets is that they are pooled from many different sources and may contain a fair amount of noise or "messiness" (e.g., Kitchin, 2014). As we have seen in the previous section, data initially often are unstructured or semi-structured rather than structured. In addition, the information can be encoded and represented in many different ways. Structured data can be distinguished into different data types, most commonly *textual* data, *numeric* data, and *date/time*-specific data. While this seems like a simple step, there are some pitfalls to be aware of in order to set up a reliable system.

First, a very basic and yet important note on *character encodings*: Textual data is always represented with a specific encoding, defining how every character of the text is technically stored as a sequence of bits. Historically, a variety of character encodings have been developed. ASCII, for example, is an early encoding standard, but only supports the 256 different characters essential to English textual data. In order to support special characters and different languages, Unicode (and the standard UTF-8 encoding in particular) has been introduced and is today dominant for online content such as news websites. When collecting data from various sources, however, it is still not uncommon to encounter deviations from this standard; before the data is stored it is therefore important to double-check the character encoding and for non-UTF-8 data to be transformed. Otherwise, special characters will break and will not be displayed appropriately, or worse: queries may overlook words that contain special characters.

For numeric data, there are different data types with implications for precision and storage size. At a basic level, *integer* types allow for numbers without decimal places and *floating point* types allow for numbers with decimal places. Each data type has a specific value range and precision, i.e. the maximum value and number of decimal places that can be stored. The higher the value range and the precision, the more space is required to store each number. A data type should be chosen carefully in order to efficiently store all possible input data. For instance, if we have a dataset of online news articles, we might want to enrich this data with information about the author of each article. The journalist's age can, for example, be restricted to a 256-value range, which equals one byte. For the number of comments on the same news article, however, 65,536 values might be needed, while article length sometimes requires an even larger type. Note that leading zeroes are not supported by numeric data types; some data such as postal codes or other IDs therefore may be needed to be stored as textual data, although they seem to be numeric in the first place.

Another common type is date and time-related data. There are data types for date only, time of day only, and timestamps, i.e. a combination of both. Besides the plain value of a date and/or time, a timestamp can also contain information about the time zone. While the latter requires more storage space, it is crucial when a research project includes international data (or plans to do so in the future); e.g. to detect that two news articles were at first glance published at the same time, but one at 3pm CET in Berlin and another one at 3pm BST in London. Depending on the dataset and the later analysis, opting for a larger data type is the safe bet to ensure a reliable information retrieval in the analysis phase.

Database schemas. The degree of certainty with which the researcher knows the expected *data types* (such as strings, integers, floats, and dates) varies depending on the *messiness* (e.g., Kitchin, 2014) as well as the *static* or *dynamic* nature of the data (see Fig. 2, IIb and c). For instance, three of the "Big Data Vs" – volume, velocity, and variety – explicitly refer to large amount of different forms Big Data can take. To take a simple example, there are hundreds of different ways of representing date and time. Thus, the more different data sources and data points we have to consider for our study, and the more our data is collected over time, the higher the chance that different schemes are used and that inconsistencies occur. In very diverse datasets, it is also quite likely that not all attributes are known for all cases, meaning that we have to deal with *missing data.* In a Big Data context, researchers have to carefully consider at which stage in their research process (as outlined in Figure 1) they want to deal with such problems.

A first approach would be to situate the handling of values that do not conform to the expected data structure as early in the chain as possible, that is before even storing them. For example, when using a SQL database, each column of the table has a clearly defined data type. If, for instance, a column expects a date in a specific format, and it receive a string that contains a date in a format that is not understood, the database can reject inserting the record (see Figure 3 for an example of the preprocessing steps necessary in such a case).

The key concept here is called a *schema*. The schema defines which properties a given object has and what kind of information these properties contain. Think of an object as an entry in the database, such as a news item. The schema would define what properties such a news item would have and of which types these properties are. A news item can have properties such as the text, a headline, an author, a date at which it was published and the number of times it was read. The schema would ensure that a news item has these properties and that the text is always a (long) character-string, the headline and author are (smaller) character-strings, the date is a date/time-object and the number of times it was read is an integer.

In other cases, it can be of advantage to deliberately decide against enforcing a specific data structure at the storage stage. A schema-free NoSQL database, for instance, might accept the insertion of values of any data type for a given key, no matter if the same key in another case refers to a value of another type: One article might contain a key called "date" with as value "2016-09-26", while the next one might have a key "date" with the value "26 sept 2016" or even 1474840800, which is the number of seconds elapsed since 1 January, 1970 – in fact a common format for representing dates. In this case, the handling of inconsistencies would have to be handled at the preprocessing or analysis stage (see Fig. 1).

Not enforcing a specific schema at storage time has a number of advantages. In particular, if data are collected on-the-fly, like in the case of continuous scraping of news sites, such an approach is more robust. If the data does not come in the expected format or if parsing fails for whatever reason, then at least *some* information is stored and the error might be restored later. This is arguably better than storing nothing. Next to this, with high-variety data, it might be hard or impossible to devise a schema beforehand, that really fits all data that are to be collected.

The downside, however, is that the later steps in the analysis pipeline have to take into consideration the possible inconsistencies in the data. This means that the researcher has to devise a system of sanity checks and fallback routines to specify how to proceed in such cases. In any event, they have to be much more cautious in interpreting the results of their analysis. The reliability of the analysis is at stake if, for instance, the researcher claims that no article was found for a specific point in time, but this was only due to the timestamp not being recognized correctly.

An interesting case in this regard is the case of social-media data. When retrieved via a an API, they usually come as an JSON object, which can directly be inserted into common NoSQL databases, which saves the researcher the effort to deal with the potentially complicated nested structure of these data: For instance, the tweets of a given user can contain a reply to another user, who in turn has some properties that are returned as well. On the other hand, especially if one is only interested in a subset of the wealth of information, one can with limited effort extract well-structured and relational information, which one could store in a schema-enforcing relational database. If one has an analysis in mind that heavily relies on the network structure of the data, one might also consider graph databases.

Taken together, in an architecture that relies on the relational database model (e.g., SQL), researchers first have to find a table structure that allows for efficient information retrieval in later phases of the research process. This means they have to consider the expected dimensions of individual tables and the combination of information across tables in advance. Conversely, database models that rely on key-value pairs instead of tables (e.g., NoSQL) have a higher flexibility and a much better scalability. Being schema-free (i.e., not enforcing a specific data structure), however, also introduces the risk that many more exceptions have to be handled in the later analysis phase, as assumptions about the data structure might not hold.

Units of data. Finally, one has to decide on what to consider a “unit” of data. This can be a file for a file-dump, a row for a relational database, a document for a document store, or a triple (a node-edge-node combination) for a graph database – all of these are databases which we will discuss in detail in a later section. For social scientists, units of data traditionally correspond to observations, typically stored as rows in a table. When analyzing news articles, tweets, or other textual data, the unit of data can also, for instance, be a document. But when questions take a different unit of analysis, such as Twitter users rather than individual tweets, or newspapers rather than articles, the unit of analysis is no longer the same as the unit being stored. Traditionally, databases excelled mainly by leveraging information about relations in data to allow these shifts to be rapidly processed.

Consider this example: We have all newspaper articles for 2016, but we are interested in the sentiment expressed toward Hillary Clinton and Donald Trump per newspaper. If we have this data stored simply as one article per file (or one big file with consecutive articles), we must go through each of these files to check whether they mention either Clinton or Trump. If they do, we then need to evaluate the sentiment expressed in the article, and figure out which outlet it belongs to. Lastly, we can update the sentiment score for the respective newspaper. As the collection of articles increases, this becomes increasingly expensive in terms of the time it takes to compute these metrics. The solution for many databases is to *normalize* the data.

Normalization implies splitting data into related units. For news articles stored in a *relational database* (such as MySQL), this may imply there are four tables: table 1 that stores an article’s ID, source, headline, content, and sentiment; table 2 that contains all named entities (each with an ID) that have been detected, among them Hillary Clinton and Donald Trump, possibly also including external information on their party affiliation or political position; table 3 that relates named entity IDs to article IDs; and table 4 that contains information on the news outlets, such as their political position. One benefit is that updating Hillary Clinton’s status from presidential candidate to nominee can be done by changing only one entry in table 2 (the ‘entities’ table), instead of changing this status for each document separately. By using this structure, a relational database can simply select the IDs of all articles that mention either Clinton or Trump without having to search through all article texts; then use this information from table 3 to select all relevant rows in table 1 – a so-called *join*-operation – and calculate the mean sentiments aggregated by source-IDs; results might lastly be added as new columns to table 4 (‘news outlets’ table).

Graph databases go even further. All information is split into small units of ID-value and ID-relation-ID pieces, so that virtually every query “joins” data. This structure is ideal for quickly sorting the information based on the relations that are relevant to a given query. This can be illustrated with the example from above: Due to the high level of abstraction that is enforced by graph databases, results can be retrieved in a single operation: We simply calculate the average of ‘sentiment-in’ related to all IDs where either ‘Hillary Clinton’ or

‘Donald Trump’ have the relation ‘occurs-in’, grouped by values of ‘news-outlet’. Because relations are stored explicitly, there is no need to look up and remember all IDs that relate information from one table to a given row in another table – we can simply look at the IDs in the ‘ID-relation-ID’ store.

If operations on the data for pre-processing or analysis require subsetting, then normalization can have significant speed gains. For instance, we might only be interested in user comments on articles from liberal news outlets. As the amount of *join* operation (i.e. getting data from different “units” together) grows, the relative advantage of a graph database’s flexible structure play out, making the system a good choice for the analysis. In the given example, this makes sense if we are interested to add information on the authors of these comments, and also analyze the ratio to which they exclusively comment within liberal news outlets.

In other cases, it can make sense to keep data together in larger units of observation (i.e *de*-normalized). In particular, as the number of *joins* as the ones described above drop, *document stores* become a better suited alternative. In systems of this type, documents are generally stored in their entirety, i.e. de-normalized. This makes data retrieval fast for different fields *within* the document, although it also means that there is much redundant information due to duplication (because shared information is stored separately in each document). For instance, the name, age, and gender of an author will be reflected in each of his articles, rather than a simple ID that refers to this information in an external table, as one single row per author. This is fast, because queries related to documents and others do not have to look them up separately and then join them: they are already contained in the same unit. However, this comes at the cost of having potentially inconsistent information: One author’s age might be recorded as 42 in the document containing one article he wrote, and as 43 in the document of another article. In contrast to a relational database, we do not have a “single point of truth”.

Comparing the different approaches to normalization by relational databases, document stores, and graph databases, Robinson, Webber, and Eifrem (2015) point to some interesting implications for dealing with very large datasets of connected data. One of them is the flexibility: In contrast to a normalized SQL database, graph database users do not have to think in advance about possible relationships, but can just “draw” new edges between their nodes as the research project progresses, just as they would draw new relationships on a whiteboard. The other one is the efficiency for some types of analyses in very large datasets. If, in our example of sentiment towards candidates, a large amount of irrelevant nodes are somewhere else in the graph, this doesn’t have a negative impact on performance: “In contrast to relational databases, where join-intensive query performance deteriorates as the dataset gets bigger, with a graph database performance tends to remain relatively constant, even as the dataset grows. [. . .] [T]he execution time for each query is proportional only to the size of the part of the graph traversed to satisfy that query, rather than the size of the overall graph.” (Robinson et al., 2015, p. 8)

III. Research design – *What’s your goal?*

Previous sections have already touched upon the various ways in which the research design guides our choice for an adequate data structure, so far mainly in an indirect manner via its obvious interconnection with a project’s data and infrastructure (see Fig. 2, III and

V). Next to this, the research design informs some considerations on a more general level, and in this way also directly impacts the choice for some database systems over others. In the following, we address some of these considerations, concerning the overall project setup, project priorities, sustainability, and data sharing.

A first consideration refers to the overall *setup* of the project. Research projects with a clear aim, scope and timeframe can opt for more rigid but powerful systems. Projects with a focus on data gathering and either no or less specific aim and scope may simply write their data to files. The *priority* in these cases is to first collect as much information as possible, and then later switch to a structured database when more specific research questions have come up. Projects with a shorter timeframe that need to quickly iterate over different approaches are likely to require flexible systems such as document stores. Such systems require less *a priori* specification and support quick modification of data-structure due to their schema-less nature. Other projects will face a self-contained dataset that can be stored in one go. In those cases, the flexibility of systems may have lower priority than the reliability and speed of retrieval that come with schema-based systems. In cases with an ongoing data collection, however, these priorities can change, as researchers need to make allowances for possible changes: When third-party datasources change their structure, schema-based systems may reject these new observations whereas schema-less systems may accept or partially-accept them. For such projects involving long-term ongoing collection of data from potentially changing sources, flexibility in storage afforded by schema-less systems may be more important than reliability concerns when analyzing data.

It should also be considered that the broader the scope of the project, and the more the data will be used in follow-up research or even for unrelated studies, the more one has to think about the *sustainability* of the data architecture. This includes issues of documentation, backup strategies, and keeping raw data – but also choosing an architecture that can be extended for new types of data whose collection might become necessary in the future. First of all, it should go without saying that regular backups – kept at different locations – are imperative. It is less clear, though, what exactly is to be backed up. Next to a dump of the database itself, it is very advisable to also keep a copy of all unprocessed raw data. For example, in a system that continuously scrapes articles or comments from an online platform, next to parsing the content and storing it in a database, it is a safe way to always also keep an independent copy of the unprocessed HTML code – maybe as a file with an incrementing number or a timestamp as filename. If, even after years, errors in the database are discovered, one can still go back to these files and still process them.

It is a good practice to always assume that others will need to access the data in the future, regardless whether *data sharing* is already a part of the project setup or not. This means that the structure of the data should be well-documented, as should be all code that is written to store and structure the data. In addition, it is very advisable to use a version control system such as *git*. On a related note, one should not assume that the data format of the database backend can be used by others in the future. Therefore, next to creating regular database dumps in the internal format the database uses, one should additionally export data to a format that might lack some of the features the database uses, but that can be read with very limited means. For instance, even though CSV files sometimes can be a pain to work with, their extremely simple structure and the fact that they are, in fact, human readable, guarantee that it will be possible to access and work with these files even in the distant future. In general, this is also true for JSON or XML files, which can be parsed relatively easily – even if their use would become uncommon. As none of these formats – CSV, JSON, XML – includes meta-information on the structure of the data, it is important to write an accompanying manual, a comprehensive readme-file, with explanations on how the data file has to be handled.

At several places before, we discussed the difference between schema-free and schema-enforcing databases. When thinking about making a database future-proof, this is also something to take into account: In the future, values might be outside of a range specified today, or it might become necessary to store additional data unavailable today. While it is difficult to analyze non-textual content today, for instance, researchers will no doubt at some point have the means to extend the Big Data analysis of news content to images or even videos.

IV. Expertise – *What are your skills?*

To some extent, the expertise required for different systems is a subjective matter: Existing expertise with systems may override concerns about system flexibility and speed; and lacking expertise may push adoption considerations towards easy-to-run systems. When building the expertise yourself is a concern, and external support such as academic collaboration or IT contracting is not available, ease-of-use becomes an important weighing factor.

The first intuition might be to save all files to disk – clearly the easiest storage solution, although it makes data exploration hard. The *relational database* model is superior in providing a solid and reliable structure. Prior to storing the documents, however, a certain expertise in designing schemas and working with relation-markup is required. Due to the fact that some a-priori decisions on the data structure cannot be reversed, relational databases are an unlikely choice for users with little to no experience and/or support. Here, the simple document-oriented nature of *document stores* shine, as they allow users to start uploading data with little to no pre-processing or structuring. The flexibility of *NoSQL* systems, such as document stores and graph databases, allows for easy changes in the data structure throughout the process, and thus requires less expertise upfront. This is especially beneficial in the *trial-and-error* process of learning how to best structure the data.

V. Infrastructure – *What else do you (plan to) use?*

Infrastructure concerns a project's interoperability with existing and future systems, including hardware and software concerns (see Fig. 2, V). When scaling is required, as is typical for Big Data projects, *hardware* becomes a concern. In the planning of the data structure, researchers should therefore already have an understanding about whether vertical or horizontal scaling is the better option based on their access to few high-power or many low/medium power machines, respectively. In simple terms, vertical scaling means getting a better computer (faster, more storage, . . .), while horizontal scaling means combining the power and storage of more simpler computers. This needs to be taken into account when designing the data architecture: Some databases are better suited for the one approach, others for the other. Especially if one cannot foresee the growth of the system in the future, having access to an infrastructure that allows for horizontal scaling can be of great value, and in the end be a key factor in deciding for one system over the other.

Regarding *software*, each of the database families comes with good, freely available solutions such as MySQL and Postgress for relational databases, MongoDB and Elasticsearch for document stores, and Neo4j or OrientDB for graph databases. What matters more is the interface between the database system and other software that is available and/or preferred by researchers. Over time, *relational databases* have built broad support among software packages, enabling most statistical environments to directly access and query the database. When working with *document stores*, interactivity is generally provided through the use of web-standards in their interface, such as JSON and, to a lesser extent, XML. *Graph databases* are relatively under-supported, which may mean that there are no interfaces available for specific software packages. If researchers decide for this system based on the many other advantages it offers, it is advisable to ensure beforehand that data access will not become a problem.

Another consideration is in how far the infrastructure supports larger cooperations. In many cases, the same dataset might be re-used for a lot of different purposes, often beyond what was foreseen at the collection stage. Next to the management of access rights within and beyond a given project, this also requires researchers to be mindful of scientific standards for data accessibility and sustainability throughout the process (see Fig. 2, III), and raises issues of privacy and data security.

Testing and assessing different data architectures

Before setting up the data architecture for a bigger project, it can be advisable to test and compare the different approaches we presented. To this end, it makes sense to use a publicly available dataset and perform a set of operations on it that are similar to what one anticipates for the own project. Such a testing necessarily depends a lot on the characteristics of the project in question, which makes it difficult to give specific instructions. A good approach is to use a publicly available datasets – like, for instance, Google's Hacker News dataset[1]. This dataset contains articles as well as comments on these articles from thehackernews.com, spanning ten years of coverage. After installing different databases of the three types discussed in this chapter and running some operations on them that resemble their specific interests, researchers can get first-hand experience in working with the dataset and compare the performance.

Conclusion and Outlook

We have shown that for many social-scientific purposes, there are no clearly 'right' or 'wrong' systems, but depending on the project at hand, the one can be more suitable then the other. Some of the most important criteria to consider are summarized in Table 1.[2]

The consequences of the choice for one system over another play out in the daily work of the social scientist and influence all other steps in the social-scientific workflow (see Fig. 1). Trade-offs can be described along two dimensions: consequences along the lines of

[1]https://cloud.google.com/bigquery/public-data/hacker-news

[2]Because file dumps are, as we have explained, mainly a useful as *additional* strategy and are very ineffective for performing actual analyses, we did not include them in the table.

Table 1
Databases and project considerations

Project considerations	Relational database	Document store	Graph database
Data	homogenous data w. little pre-processing	heterogenous data w. much pre-processing	semi-structured data
Research design	clear, narrow questions with fixed datasets or data with fixed structure	exploring but unit-centered, ongoing third-party datasets	broad, relation-focused with varying data
Expertise	legacy software requires training prior to data collection	quick development with low barriers to entry	specific skills but relatively simple model
Infrastructure	broad existing support in software libraries for inter-operability	easy integration in modern packages and machines based on web-standards (JSON); easy horizontal scaling	may require custom integrations and adaptations

ease-of-storage versus reliability-of-retrieval on the one hand, and consequences along the lines of ease-of-use versus power-to-explore on the other.

Ease-of-storage vs. reliability-of-retrieval. Ease-of-storage is strongest in *document store*-systems such as MongoDB. These databases do not enforce a schema and will accept new units of data even when the data-types contradict earlier data-types. Similarly, *graph databases* generally do not enforce a schema, so that added nodes and edges can have keys with values of varying datatypes. The flexibility of these systems comes at the expense of the reliability of retrieval. Take the case of ten documents with a field called "created_at", of which six are stored as date/time-data types and four as strings (e. g. "29 feb 2016"). A query that looks for all documents in a specific time range will only retrieve the six entries with the correct format and most likely silently ignore the four that come in another, as they cannot be processed. The same problem occurs with arithmetic operation on data where a string is stored but an integer or float is expected. With such schema-less systems, researchers must be careful to check their assumptions about data types.

Reliability of retrieval is highest in *relational databases* such as MySQL. By enforcing a schema, these databases ensure that queries provide an appropriate response over all data. The downside of enforcing a schema is that the ease-of-storage suffers. For instance, when data is actively gathered by means of a web scraper, an inconsistent date string (e.g., "02/29/2017" where "29-2-2017" is expected) might mean it is no longer accepted by the database. The database might throw an error, or simply not store any information. Some common problems that must be handled by the researcher are data extraction from strings (such as "#10" to 10), recoding of missing values ("none" to Null) and unexpected new fields. It is important to understand that such observations – albeit not schema-conformant and therefore in need of pre-processing – are still valid and important in terms of research purposes. Researchers opting for schema-based systems need to keep this in mind and be careful to avoid the loss of information. A reliable workflow therefore starts by creating a backup in parallel, e.g. by gathering the data into file-dumps. From there, the information still can be transferred into the database, allowing researchers to later restore otherwise lost information.

Ease-of-use vs. power-to-explore. Another difference in the consequences for the researchers' daily work may lie in the focus on easy-to-use systems over power-to-explore. *Document stores* are generally oriented towards storing data with little to no preparation of the system, whereas *graph databases* are pointless without the definition of relations and *relational-databases* even require careful planning and specification of a schema before inserting any data. The costs of these pre-storage decisions comes in both the training needed to understand the choices and their repercussions, as well as in designing and applying these data-models to the database in the context of a given project. The upside of using systems with an explicit data-model go beyond the assurances about reliability-of-retrieval. As a trade-off for requiring more expertise and planning, researchers are awarded with a generally more powerful system in terms of exploration-affordances. *Relational databases* shine in their ability to aggregate at different units-of-analysis due to the normalized nature of the data. By spreading observations in different units of storage, relational databases can quickly examine relationships in the data. Another advantage is that computations and enrichment can be added efficiently; for instance, to add the gender of a journalist to the database, only the respective entry in the author table needs to be edited, rather than going through all news articles to add the information there. *Graph databases* provide even more support for exploring a dataset by fully reducing data to node-edge-node triples: By linking individual values through relations, queries such as "how many articles do which authors write about the elections?" are easy to compute – provided that the articles have previously been enriched with a key that stores information on their topic.

Naturally, the considerations presented in this article are not exhaustive. In particular, we did not discuss the possibility of scaling up an infrastructure using frameworks such as *Hadoop*, in which both calculation and storage of large data sets are distributed across multiple machines in a cluster. The strength of MapReduce based solutions such as Hadoop is that they distribute a task across several machines. This is mainly interesting for the analysis of Big Data, as it greatly speeds up the analysis. But it also implies the distribution of data using a distributed file system. In that sense, the choice for a framework like Hadoop is related to data architecture as well. However, this does not necessarily have to result in fundamentally different choices: Databases like MongoDB and Elasticsearch can simply be run on top of Hadoop. Horizontal scaling using such frameworks works very well for most *document stores*, which means one can easily add additional machines to the system if necessary. Scaling up a *relational database* system can be more tricky. For many social-scientific questions, systems that run on just one machine are sufficient, especially for applications in which performance plays a role or in which a lot of data is created dynamically. If horizontal scaling is needed, however, the general decision framework we presented here (see Fig. 2) is applicable to such contexts as well.

In this chapter, we outlined how the amount and characteristics of data often used in computational social science influence the traditional social-scientific research process.

In particular, we challenged the implicit assumption that data storage is mere logistics, without any meaningful considerations for researchers to make at this stage. We state that, in computational social science, the choice for a specific data architecture has become an independent step in the research process. Although mundane, its significance can be compared to the way the sampling or the choice for specific instruments and levels of measurement shape the (traditional) research process in terms of the analysis that can be conducted later on. Similarly, Vis (2013) argued that when collecting online data, the researchers' choices and the characteristics of the techniques used (e.g., APIs) "make" the data. She writes: "Researchers should aim to make themselves more aware and reflect more on the process through which they have collected data and make this as transparent as possible." We want to extend this line of reasoning to the data architecture and the choices researchers make around the storage of the data. The misperception of this stage as a mere technical problem reflects an outdated research logic that is no longer sufficient in the age of Big Data.

Considering this, we proposed four dimensions that need to be addressed in Big Data projects to assist researchers in their choice for an adequate system: the data, the research design, the expertise, and the infrastructure. We hope that the proposed guidelines help to spark a sound methodological discussion about the research process in our field, that does not see questions of data storage as a merely technical problem, but as an integral part of the computational social science.

References

Behnel, S., Faassen, M., & Bicking, I. (2016). *lxml – XML and HTML with Python* (Tech. Rep.). Retrieved from http://lxml.de/

Boumans, J. W., & Trilling, D. (2016). Taking stock of the toolkit: An overview of relevant autmated content analysis approaches and techniques for digital journalism scholars. *Digital Journalism*, *4*(1), 8–23. doi: 10.1080/21670811.2015.1096598

Bryman, A. (2016). *Social research methods* (5th ed.). New York, NY: Oxford University Press.

Burscher, B., Odijk, D., Vliegenthart, R., de Rijke, M., & de Vreese, C. H. (2014). Teaching the computer to code frames in news: Comparing two supervised machine learning approaches to frame analysis. *Communication Methods and Measures*, *8*(3), 190–206. doi: 10.1080/19312458.2014.937527

Comella-Dorda, S., Lewis, G. A., Place, P., Plakosh, D., & Searcord, R. C. (2001). *An enterprise information system data architecture guide* (Tech. Rep. No. CMU/SEI-2001-TR-018). Retrieved from http://repository.cmu.edu/cgi/viewcontent.cgi?article=1136&context=sei

Field, A. (2016). *An adventure in statistics: The reality enigma.* London, UK: Sage.

Grimmer, J., & Stewart, B. M. (2013). Text as data: The promise and pitfalls of automatic content analysis methods for political texts. *Political Analysis*, *21*(3), 267–297. doi: 10.1093/pan/mps028

Günther, E., & Quandt, T. (2016). Word counts and topic models. *Digital Journalism*, *4*(1), 75–88. doi: 10.1080/21670811.2015.1093270

Günther, E., & Scharkow, M. (2014). Automatisierte Datenbereinigung bei Inhalts- und Linkanalysen von Online-Nachrichten. In K. Sommer, M. Wettstein, W. Wirth, & J. Matthes (Eds.), *Automatisierung in der Inhaltsanalyse* (pp. 111–126). Cologne, Germany: Herbert von Halem.

Inmon, W., & Linstedt, D. (2014). *Data architecture: A primer for the data scientist: Big Data, data warehouse and data vault.* Waltham, MA: Morgan Kaufmann.

Jacobi, C., van Atteveldt, W., & Welbers, K. (2016). Quantitative analysis of large amounts of journalistic texts using topic modelling. *Digital Journalism*, *4*(1), 89–106. doi: 10.1080/21670811.2015.1093271

Kitchin, R. (2014). Big Data, new epistemologies and paradigm shifts. *Big Data & Society*, *1*(1), 1–12. doi: 10.1177/2053951714528481

Kitchin, R., & McArdle, G. (2016). What makes Big Data, Big Data? Exploring the ontological characteristics of 26 datasets. *Big Data & Society*, *3*(1), 1–10. doi: 10.1177/2053951716631130

Kohlschütter, C., Fankhauser, P., & Nejdl, W. (2010). Boilerplate detection using shallow text features. In *Proceedings of the third ACM international conference on web search and data mining* (pp. 441–450).

Maireder, A., Ausserhofer, J., Schumann, C., & Taddicken, M. (Eds.). (2015). *Digitale Methoden in der Kommunikationswissenschaft.* Berlin. doi: 10.17174/dcr.v2.0

Pomikálek, J. (2011). *Removing boilerplate and duplicate content from web corpora* (Doctoral dissertation, Masaryk University). Retrieved from http://is.muni.cz/th/45523/fi_d/phdthesis.pdf

Richardson, L. (2015). *Beautiful soup documentation* (Tech. Rep.). Retrieved from https://www.crummy.com/software/BeautifulSoup/

Robinson, I., Webber, J., & Eifrem, E. (2015). *Graph databases: New opportunities for connected data.* Beijing: O'Reilly.

Trilling, D. (2014). Weg vom manuellen Speichern: Automatisierte Datenerhebung bei Onlinemedien. In K. Sommer, M. Wettstein, W. Wirth, & J. Matthes (Eds.), *Automatisierung in der Inhaltsanalyse* (pp. 743–789). Cologne, Germany: Herbert von Halem.

Tupper, C. (2011). *Data architecture: From zen to reality.* Waltham, MA: Morgan Kaufmann.

Vis, F. (2013). A critical reflection on Big Data: Considering APIs, researchers and tools as data makers. *First Monday*, *18*(10), 1–16. doi: 10.5210/fm.v18i10.4878

Waldherr, A., Maier, D., Miltner, P., & Günther, E. (2016). Big Data, big noise: The challenge of finding issue networks on the web. *Social Science Computer Review*, 1–17. doi: 10.1177/0894439316643050

Appendix: Overview of Technical Terms

Term	What is it?	Example
HTML tag	Building block of web pages auch as title, heading, paragraph, hyperlink, image, list, table	`<a href="http://www.nytimes.com/">The New York Times</a>`
CSS	Cascading Style Sheets; language to describe the presentation of a web page; e. g. attributes style classes to HTML tags that indicate presentation such as blue color	`<a href="http://www.nytimes.com/" class="link_to_source">The New York Times</a>`
CSS selector	Rule to find and extract a specific part of a web page, looking for a tag with a specific CSS class	`class="link_to_source"` could be targeted to extract all blue hyperlinks in a web page
XPATH	Rule to find and extract a specific part of a web page, similar to CSS selectors	

(Information) Retrieval	Describes the activity of providing access to unstructured information in various data storage systems	
Data-related		
Data	A collection of observations	newspaper articles, posts on Hacker News, etc.
Unit of data / unit of observation	a set of data-points, generally about one object, such as a newspaper article, which includes data points such as the headline, the content, the author, the number of times read etcetera.	A newspaper article, a post on Hacker News
Data points	a specific value associated with a unit of data, such as the name of an author of the number of times an article is read	10 (times read), "John Doe" (author)
Data model	An implicit, but preferably explicit understanding of data including the format and relations. Not to be confused with the schema, which is a specific formatting requirement based on the more abstract data-model	
Database-related		
Schema	Describes the structure of a database, i.e. which tables exist with with columns and how they are related to each other, often represented as a diagram	
SQL	Structured Query Language; language to interact with a relational database; typically used to insert, modify, query, and delete data	
SQL query	is an operation based on the SQL language	Find the titles of all articles from the source with id 5: `SELECT title FROM articles WHERE source_id = 5;`

SQL join	Combination of two tables in an SQL query based on a common field between them	Find the titles of all articles including URLs from the source with name “The New York Times”: `SELECT title, url FROM articles JOIN source ON articles.source_id = source.id WHERE source.name = ‘The New York Times’;`
Document	Unit-of-storage for document-stores, often a variant of JSON	(see JSON)
JSON	JavaScript Object Notation, a common web-format for data interchange	{"headline":"How to store data", "author":"John Doe", "tags":["databases","tutorial"], "times_read":10}
Triple	a value-relation-value set stored in a graph database / triplestore	"John Doe"–"author of"–"How to use databases"