An Empirical Study of Open Data JSON Files

Mark Lukas Möller
University of Rostock
Germany
mark.moeller3@uni-rostock.de

Nic Scharlau
University of Rostock
Germany
nic.scharlau@uni-rostock.de

Meike Klettke
University of Rostock
Germany
meike.klettke@uni-rostock.de

ABSTRACT

JSON is challenging XML as lingua franca of data exchange formats. Likewise, the JSON schema initiative has been gaining momentum to the point where it is considered by many as a defacto standard. Both the JSON data format and the JSON schema formalism allow for great degrees of freedom in modelling nested and semistructured data. In this article, we introduce a scalable tool, jHound for profiling JSON document collections. We use this tool to pursue the question how real-life datasets are structured, and whether developers actually make use of the features offered by these languages when modelling their data. For this analysis, we focus on JSON documents available as open data. While this sample can only deliver a biased snapshot of what real JSON is like, we gain first insights into the common practices in modelling data with JSON. The jHound tool is provided and available as open source, so that scientist and practicioners can apply it for profiling of other JSON data sources.

KEYWORDS

NoSQL Data Exploration, Metrics for JSON Data, JSON Data Profiling, Open Data Analysis

1 INTRODUCTION

Data for data science applications nowadays are provided in various formats. Besides the established relational databases, various data sources are available as NoSQL data, especially in JSON format, a very popular data format that was originally developed for data exchange between web applications. NoSQL databases are an alternative to traditional relational databases when scalable database solutions for storing large quantities of data are needed. In these lightweight NoSQL databases systems data with an arbitrary internal structure can be stored and in most NoSQL systems it is not required to define a schema before storing data.

If such heterogeneous NoSQL data shall be used in data science applications, an ETL process is necessary for transforming these data into a certain structured target format as shown in Figure 1. The first step in this process is understanding of the datasets. For this we have to determine the statistics and commonalities of JSON files and need the opportunity for a comprehensive data exploration. Users must be able to explore and understand the structure of the data set in depth, know the data types, the completeness of the data (e.g. percentage of null values in each property), the regularity of the data sets, the nesting depth, and their overall size. The derivation of several metrics describing these characteristics is represented in this article.

In short, the exploration of NoSQL data is always the first step in all data engineering tasks in data science. With *jHound*, we introduce a method for *analyzing and using available JSON data* and provide a efficient *tool for JSON data exploration*. Furthermore, we apply the approach onto available data sets to understand how JSON is used in different applications. We use open data repositories to test our techniques, calculate statistics and find interesting patterns in data, such as relationally stored data dumped into the JSON data exchange format. In this article, we examine how the capabilities of JSON are used. Is it common that data is deeply nested? Do people use optional properties or do they store data in a traditional regular way?

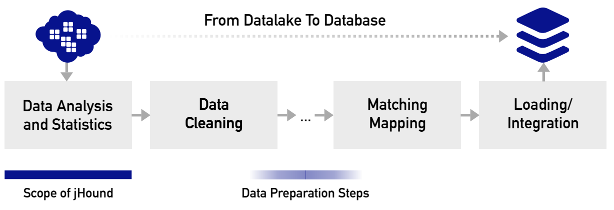


Figure 1: Pipeline from a datalake to a database

Outline. The rest of the article is structured as follows: In Section 2, we introduce the tool jHound and give a performance evaluation of the parallel implementation. Section 3 focuses on the results of selected parts of the CKAN data profiling. In the next section, related work in the fields of data profiling and schema extraction is given. Finally, it is shown how jHound can be applied for analyzing own JSON data in Data Science projects.

2 THE JHOUND ANALYSIS TOOL

jHound[1] has been developed for data exploration of NoSQL datasets. It scrapes links from open data repositories, downloads the NoSQL datasets, parses them and derives several metrics from the data. jHound is written in Python as a distributed system with a map-reduce process using multiple analysis servers, referred to as *nodes*.

To learn more about the variety of JSON data and the way how JSON is utilised, we use the *CKAN* repositories for our analyses. CKAN is a project in which any kind of public data can be stored, often used by governments to provide access to public information. Hence, CKAN repositories are a good baseline for the retrieval of JSON documents because it contains a wide variety of different kinds of JSON data. CKAN consists of packages, which, in itself, consist of URLs to documents, called resources. Every repository, every package, and every resource comes with some metadata such as the total amount of elements, data types, and more. For all data sources, the jHound analysis workflow consists of the following three steps: Retrieval, Analysis, and Visualization.

2.1 Data Retrieval

JSON Documents are fetched via CKAN's RESTful API while the URLs to resources with a specified format of either `JSON` or `GeoJSON` are stored in our database. Afterwards, the files behind the crawled URLs are downloaded simultaneously on multiple machines sharing a network-based storage. We analyzed a total

[1]Source code: https://jhound.de, documentation: https://docs.jhound.de.

Abbrev.	Provider	Fetched at	# Scraped	# Analyzed
DHC	datahub.ckan.io	2020-08-23 20:30:09 CEST	537	197
HRO	opendata-hro.de	2020-08-23 17:44:41 CEST	386	382
DRP	dados.recife.pe.gov.br	2020-08-23 17:43:44 CEST	282	280
CPS	ckan.publishing.service.gov.uk	2020-08-20 11:54:36 CEST	4,988	500
ZUE	data.stadt-zuerich.ch	2020-08-15 19:30:46 CEST	151	14
GOV	data.gov.ie	2020-08-15 19:30:17 CEST	1,604	1,494
BOD	data.bodik.jp	2020-08-15 19:27:30 CEST	108	107
HDX	data.humdata.org	2020-08-15 19:21:39 CEST	1,036	712

Figure 2: The JSON data sample

of 3,686 JSON data documents. This includes data published by the government of Ireland, open data portals from cities such as Rostock [HRO] or Zurich [ZUE], NGOs such as Humanitarian Data Exchange [HDX], and others. The sizes of individual files range from less than 16 B to well over 1 GB. The table in Figure 2 describes the data sets by introducing an abbreviation, stating the web source, the number of scraped files, and the number of successfully analyzed files.

2.2 JSON Data Analysis

The analysis step treats the downloaded JSON documents as input information. Making use of *ijson*, an iterative, event-driven JSON parser, all JSON documents represented as trees are analyzed. Here, the JSON structure is interpreted by ijson as triples, whereby each triple consists of a prefix, an event (e.g. for an observed begin of an object), and a respecting value. jHound collects the following document metrics during the analysis process which are relevant for this article's key research question to understand how actual JSON documents look like:

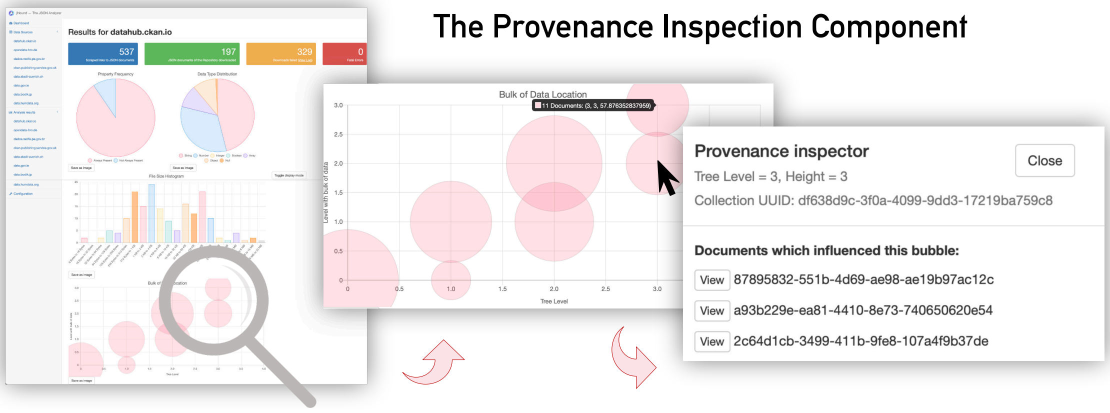
- **Document Tree Level (DTL)**. This metric measures how deep documents are nested. If there is a JSON document which only consists of a single object with one or a set of properties which are not nested, DTL is 0. If there are objects with properties whose values represent other objects, the DTL increases by one per object nesting level.[2]
- **Bulk of Data Level (BDL)**. The bulk of data level represents the information on which DTL most properties can be found. We often faced documents having their bulk of data not on the root level which shows that nesting capabilities of JSON are actually used.
- **Data Type Inference and Distribution**. The JSON syntax [4] allows to distinguish between different data types. The jHound tool keeps track of the used data types for all JSON documents. According to the definition of JSON, we distinguish between the following data types of the property values: *object*, *array*, *string*, *integer*, *number* (non-integer numeric values), *boolean*, and *null* values.
  In addition to the analysis with ijson, jHound tries to parse each string to a float to check whether the string has been "abused" to represent a numerical value. If a value of a string is either "true" or "false", representing a boolean value, we register this kind of abuse as well. Such typecast can be found in several JSON documents namely those which were presumably transformed from other data formats into JSON. Furthermore, we check documents for empty strings to get an insight whether the possibilities of optional properties or null values are used.
- **Property Occurrence**. The property occurrence metrics provides an insight of potentially required and optional attributes. This is a metric inspired by JSON schema which is able to distinguish between required and non-required properties. We infer a similar metric from the documents but do not use JSON schema since this schemas are not included in the scraped files. Interpreting nested JSON documents as a tree, the properties of each node are collected per nesting level. Afterwards it is checked if a certain property is available in each node of the respecting level. If this is the case, we treat this property as required and otherwise as an optional property. Another interpretation is "always available"/"not always available". This metric evaluates the regularity or heterogeneity of the JSON documents. It is possible to have valid JSON documents where both the required and the non-required count is zero, e.g. if the document only consists of an array of values without keys. Object-valued properties in arrays without a specified key are regarded as optional values, too.
- **Document Metadata**. We collect some metadata about the JSON documents, e.g. their filesizes and origins. Together with the raw analysis data about the previously mentioned metrics, we want to see if there are similar JSON documents which might belong together and are stored in chunks in the repositories. Additionally, metadata helps to find out whether documents represent specialized JSON, such as GeoJSON, by looking for predestinated keys and properties.
- **Repository Metadata**. CKAN repositories allow to host arbitrary types of data. Regarding the repository curation, we inspect the metadata of the repositories to find out how many documents are mislabelled as JSON. This helps to estimate if data can be analyzed on-the-fly or if another data preprocessing step has to be made before analyses.

Additionally, we investigated how well jHound scales. For this, we analyzed all repositories on one local node with only one thread, as well as on multiple nodes with four threads each. We determined that threading was the best way to speed up the JSON analysis process. As apparent in Figure 4, the speed up factor between one thread and two nodes with eight threads in total reduced the analysis time by the factor 8.24 in the best case (Figure 4, CPS) and by the factor 3.75 in the worst case (Figure 4, DRP). If additional nodes are used for the analysis, it is apparent that there is no (e.g Figure 4, DRP) or only a slight speedup (e.g Figure 4, CPS: Factor 1.16 for two versus three nodes and factor 1.15 for three versus four nodes). In comparison to the increase of threads per node, the speed up is negligible. The reason for this can probably be attributed to jHound's map-reduce internals and its data store which we plan to optimize in the future. In general, it is possible to add or remove analysis nodes on demand during a running analysis. However, the number of nodes and threads was static during all analysis processes.

2.3 Visualization

Finally, the results of the analysis process are shown in raw data as well as in certain diagrams. Figure 3 gives a first impression

[2]While the ECMA 404 specification and the JSON RFC do not specify how *depths* is defined, we rely on definition e.g. from IBM where only the depths of objects is taken into consideration, while arrays are not (c.f. https://www.ibm.com/support/knowledgecenter/SS9H2Y_10.0/com.ibm.dp.doc/json_maxnestingdepth.html [2021-01-21], https://tools.ietf.org/html/rfc7159 [2021-01-21]).


The Provenance Inspection Component
Results for datahub.ckan.io
Bulk of Data Location
Tree Level
Level with bulk of data

Provenance inspector

Close

Tree Level = 3, Height = 3

Collection UUID: df638d9c-3f0a-4099-9dd3-17219ba759c8

Documents which influenced this bubble:

View 87895832-551b-4d69-ae98-ae19b97ac12c

View a93b229e-ea81-4410-8e73-740650620e54

View 2c64d1cb-3499-411b-9fe8-107a4f9b37de

Figure 3: The provenance component for the inspection of the main data location levels

Repository	Local analysis, 1 thread	Distributed analysis, 4 threads per node 2 nodes	3 nodes	4 nodes)
DHC	156.39 s	35.20 s	34.03 s	29.96 s
HRO	2130.20 s	364.14 s	308.65 s	312.50 s
DRP	540.30 s	144.03 s	146.97 s	146.32 s
CPS	38 459.47 s	4664.00 s	3992.94 s	3450.45 s
ZUE	7.17 s	2.06 s	1.93 s	1.64 s
GOV	15 163.72 s	2500.89 s	1921.17 s	1717.58 s
BOD	385.84 s	59.19 s	50.49 s	46.42 s
HDX	6032.33 s	1160.41 s	1098.58 s	1041.11 s
Total	62 875.42 s	8929.92 s	7554.76 s	6745.98 s

Figure 4: Average runtimes with respect to the amount of nodes and threads used for the analyses

on the results overview and provenance component of the user interface of jHound.

The graphical visualizations provide a quick overview on certain JSON document characteristics and so support the data exploration tasks. Additionally, a user can use a provenance component to see which documents influenced which part of the result.

3 KEY INSIGHTS IN OPEN DATA JSON FILES

There are numerous characteristics of JSON documents that influence how the documents are processed. The jHound exploration tool is focused on the most important metrics: tree characteristics like size and depth, distribution of data types, nesting, and so on. In this chapter, we describe these metrics for selected JSON documents from the CKAN repository which mainly are:

- We inspect the metrics of the JSON files and show which conclusions can be drawn from the metrics to determine necessary subtasks in the data preprocessing and what information about the usability of the data can be derived from the metrics? What is the distribution of data types and how often are they abused? Is there a significant amount of specialized JSON, such as GeoJSON?
- We question how intensively relational and non-relational approaches are used. We want to find out if data is stored flatly or if the capability of nesting data is used. If so, up to which level? Is it possible to find documents which may belong together by using our metrics?
- The last question deals with how well data is curated in open data repositories. When downloading data from a repository, which potential problems occur during an on-the-fly analysis process? How many files are labeled as JSON but actually contain XML or HTML?

To answer the first questions, we investigate the metrics introduced in Section 2 and got the following results.

Required vs. non-required properties. Across all 3,686 analyzed datasets, approximately 66% of the properties are required properties while the rest are optional. This implies that documents often follow a kind of regular structure. Yet, the occurrence of completely unstructured documents was very rare. Often, we find a fully regular structure which probably results from the use of programs which either require a certain structure to be parseable or which generate the JSON documents in a regular structure.

Consequences for data imputation and data analysis: The information about required and optional properties defines a measure for regularity of datasets, which is required for further data processing, e.g. integrating the data into relational storage environments such as RDBMS systems. This information can also be used for the decision which datasets are selected because it reflects the completeness of information. With the jHound tool this is clearly displayed at a glance.

Distribution of datatypes. Across all repositories and documents, more than 2.68 billion properties were analyzed. By far the most frequent data type in the collection are numbers with over 50% of occurrence, followed by arrays and strings. Only 1.12% of properties are named objects containing other properties or other objects. The least common data types are booleans and null values. We did not expect arrays to occure with such a high frequency and took a manual look at the documents. We determined that this is often caused by GeoJSON-documents which encode tuples of coordinates as arrays. Figure 5 shows the distribution of JSON datatypes for the CKAN excerpt. If the data is integrated into a relational database schema, a lot of 1:n relationships occur due to the numerous arrays, which have to be taken into account with regard to performance, especially when reconstructing by using join operations. If numerical operations like aggregations are in the foreground of the subsequent analysis, a column-oriented design should be adopted as a result of the predominance of numbers.

Nesting characteristics. Interesting insights came up when we correlate this type distribution with our analysis where most of the data resides when interpreting the JSON documents as a tree. As apparent in Figure 6, most of the properties reside in tree level 2 (starting with level 0 for "flat" documents). Even though objects make up 1.12 % (Figure 5), most of the data is located in nested

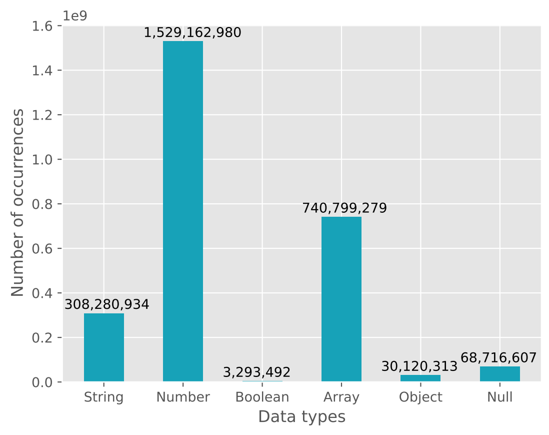


Figure 5: Datatype distribution across all datasets

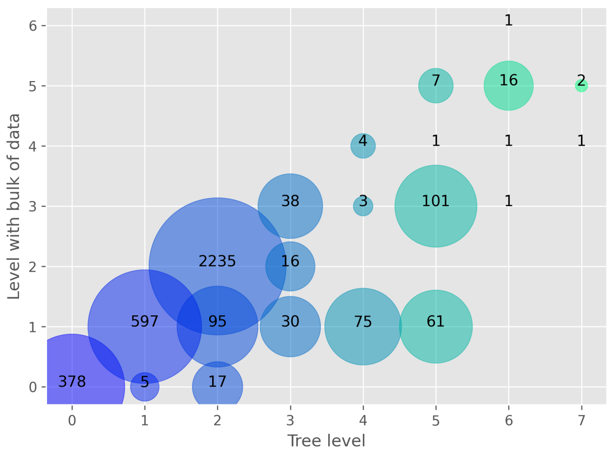


Figure 6: Connection between the DTL and the BDL of all analyzed documents

data structures. 2,235 of 3,210 analyzed documents, have a DTL of 2 whereby most of the data resides in tree level 2 (BDL 2), followed by 597 documents with a DTL of 1 with a BDL of 1, followed by 378 documents with a DTL and BDL of 0. When representing the connection between the DTL and BDL in a diagram, it is apparent that if JSON documents are nested, the bulk of data resides mostly in the deepest tree level. Having a look at all documents, the most deeply nested documents found had a tree level of 7 whereby most of the data resided on level 5 in these documents. *Consequences for Data Transformation:* The nesting information is important if JSON documents have to be transformed into relational databases. Beside that, the nesting depth is a hint that reveals the data model complexity and processing costs. Especially in systems where path operations are expensive to execute with regards to the system's resources, data scientists may unnest the documents before post processing the data.

Geographical data. We inspected the documents for certain keywords which are typical indicators for GeoJSON. They typically consist of properties with keys such as *FeatureCollection*, or *MultiPolygon*. We found out that 466 of 3,686 – this is about every eighth document – contained at least one of those properties and were classified as GeoJSON. Considering this fact, it can be assumed that GeoJSON as a special case of JSON plays more than a minor role in applications. As a consequence, jHound's metrics can be used to spot GeoJSON data, such as by certain keywords or pairs of coordinates inside of arrays. This is a valuable aid if a geospatial database is to be built based on the analyzed data.

Using metrics for classifying datasets. Taking raw metrics into account also supports answering the questions how intensively relational vs. non-relational approaches are used. For example, in one repository, we explored that jHound found more than 100 documents with a very similar metrics pattern. Having a manual inspection of these documents revealed that these documents semantically belonged together. If we find JSON documents with an object nesting depth of 0, it is a very strong indication of relational data exported as JSON documents into arrays.

Consequences for data ingestion. Hence, we can conclude from the jHound analysis metrics as well as from the manually inspected files with the same metrics pattern that it is not uncommon that relational data is stored in JSON and documents in the same repository have a cross-document connection.

It can be inferred that similar values in certain metrics are an indicator for similarity of dataset structures. Due to this, metrics can be used for identifying datasets with certain characteristics and select these datases for data ingestion.

Data quality. Another question of interest is the evaluation of *data quality*. JSON supports multiple data types as shown, but from our own experience, we know of the existence of low-capability tools which store any information in the most general data type, a string. Technically, this works, but may be hard to interpret for other tools when reading this data because type-casting must be executed beforehand. Therefore, we took a look if strings were abused in this way. We distinguish between the following types of abuse and classify these in the following:

- **Boolean Abuse.** Since JSON supports booleans, we expect the use of property values like True and False. However, we often found them encoded as strings.
- **Numerical Abuse.** Analogous to the boolean abuse, we had a look how often numerical values are stringified.
- **Empty String Abuse.** Because JSON allows optional properties, we typically expect that properties with an empty value are omitted.

We found out that a third of all documents contained empty strings as shown in Figure 7. We suspect that data is often stored regulary when exported by systems which generated those documents and therefore, empty values are used for representing null values. However, the metrics must be interpreted with caution since empty strings might also have a semantical meaning. The same fact holds for numbers encoded as strings. The encoding of booleans as strings is rather rare. We found only 73 documents where this was inspectable. This is analogous to the rare use of the data type boolean in the CKAN excerpt, which could be seen in Figure 5. In any case, if datasets which contain such type deriavtions shall be processed, type castings have to be executed as part of the data preprocessing step.

Data Repository Curation. jHound also offers the opportunity to check link constraints, We also investigated the curation quality of the analyzed open data repositories and their aptitude for an ad-hoc analysis of open JSON data. Across all repositories, we scraped links to 9,092 JSON documents. Merely 3,676 or 40.43% of the documents were suitable for the analysis by our tool. For the other 60% of the documents, our analysis process resulted in an

Abuse Type	# Affected Documents	# Affected Properties
Boolean	73	4,337
Number	1,121	55,932,374
Empty String	1,076	12,286,977
Total	**2,270**	**68,223,688**

Figure 7: Datatype abuse by documents and properties

erroneous state. Regarding the huge amount of failed documents, the most influencing fact is probably the CPS repository where only 500 documents could be downloaded while the other ones fail with no server response. Ignoring this repository results in a scrape/analysis success rate of 78%. jHound tracks occurring errors to notify where the workflow fails. There can be different reasons for link errors: ranging from malformed documents to server side rate limit which cuts our connection. The most frequent errors jHound found were malformed documents and HTTP status codes 404 or 403, indicating that the repositories either have "dead links" to non-existing JSON documents or are in network areas that exclude public access. For repositories such as HDX, we often encountered the HTTP error 429 which indicates that jHound made too many requests in a specific time period and was therefore prevented to download all JSON files.

An investigated fact regarding data quality and repository curation is that *CKAN* repositories often announce datasets in wrong data formats, for example the ZUE repository. Here, errors mainly occurred with error code *"Parsing after download (Unexpected symbol '<' at 0)"*, implying other data formats like HTML or XML, although if the file was listed as JSON. Because this error occurred for more than 100 files, we inspected the repository manually. Unfortunately we found out that the repository curators labeled the resources as JSON, but instead of providing a downloadable file as expected, the JSON file is embedded in an online map and serves for the display of coordinates[3]. Hence jHound finds an uninterpreteable web page. While our tool is able to report unparsable documents and report the underlying cause, we did not inspect broken documents further since the structural and semantic meaning cannot be recovered.

Taking all the results into account, we conclude that a naive on-the-fly analysis is not possible. Not all repositories are perfectly curated and often, links are not existing anymore or do not provide a way to directly access the JSON documents. Therefore, a variety of pre-analysis steps to mitigate errors are required. This also holds for the JSON documents themselves. Often, non-string data types are encoded as strings and require a typecasting before processing. The jHound tool allows to discover these potential problems regarding data quality both for own and openly hosted data. It gives a first, brief insight how data looks like and points out typical pitfalls which have to be solved before the data is processed in the following data cleaning step (c.f. Figure 1).

4 RELATED WORK

In the last decades, different data metrics have been developed for different data formats, Document depth and the bulk of data metrics were introduced in [6]. Byron Choi published a well-known article entitled "What Are Real DTDs Like" [2]. In this article it was examined whether all features that are offered by DTDs for defining the structure of XML documents are used in real applications. Our work is also related to the ongoing efforts in extracting a schema from large collections of JSON data, e.g. [1, 3, 5, 7, 8, 10]. All these metrics influenced the development of the jHound approach. The first version of this tool [9] included basic metrics like the distribution of data types as nesting depths. Since then, the current version now implements a parallelized workflow with the ability to add or remove analysis nodes on-demand and during analyses. Additional metrics such as the incorrect usages of data types were added to the inspection as well, which supports the data cleaning process as given in Figure 1.

In our own work, the data exploration task is embedded in a schema evolution and migration approach for NoSQL databases where a NoSQL schema evolution language, different efficient data migration strategies and a migration advisor[4] are developed for this purpose.

5 CONCLUSION

For this article we analyzed open JSON data and present our results and a brief insight was given into jHound, our JSON analysis tool, which implements a full pipeline for scaping, downloading, and analyzing JSON documents from open data repositories. We inspected 8 repositories, among them ones of governments and cities, to find out how JSON is used. We determined that not all data repositories are well curated and often, a preprocessing step is required in advance of the analysis. During the analysis of JSON documents, we have shown that often JSON is used with all its capabilities such as optional properties and nested data, while we also found documents that seemed to be generated and exported from origins storing relational data. We explained the challenges like abused data types. With jHound we were able to use JSON metrics pattern to find documents which belong together in the same repository. In the future, we plan to automate this process and aim to inspect the JSON documents more in-depth with different metrics.

Acknowledgements. The article is published in the scope of the project *"NoSQL Schema Evolution und Big Data Migration at Scale"* which is funded by the *Deutsche Forschungsgemeinschaft (DFG)* under the grant no. 385808805. A special thanks goes to Nicolas Berton, a former guest student of the University of Rostock, whose ideas and whose first draft of the jHound tool largely influenced the outcome of this article.

REFERENCES

[1] Mohamed Amine Baazizi, Houssem Ben Lahmar, Dario Colazzo, et al. 2017. Schema Inference for Massive JSON Datasets. In *Proc. of the 20th Intl. Conf. on Extending Database Technology, EDBT 2017.* OpenProceedings.org, 222–233.
[2] Byron Choi. 2002. What are real DTDs like?. In *Proc. WebDB 2002.* 43–48.
[3] Michael DiScala and Daniel J. Abadi. 2016. Automatic Generation of Normalized Relational Schemas from Nested Key-Value Data. In *Proc. SIGMOD 2016.* ACM, New York, 295–310.
[4] Ecma International. 2017. The JSON Data Interchange Format. Standard ECMA-404, 2nd Edition.
[5] Angelo Augusto Frozza, Ronaldo dos Santos Mello, and Felipe de Souza da Costa. 2018. An Approach for Schema Extraction of JSON and Extended JSON Document Collections. In *2018 IEEE Intl. Conf. on Information Reuse and Integration (IRI).* IEEE, 356–363.
[6] Paola Gómez, Claudia Roncancio, and Rubby Casallas. 2018. Towards Quality Analysis for Document Oriented Bases. In *Conceptual Modeling.* Springer International Publishing, Cham, 200–216.
[7] Javier Luis Cánovas Izquierdo and Jordi Cabot. 2013. Discovering Implicit Schemas in JSON Data. In *Web Engineering - 13th Intl. Conf., ICWE 2013. Proc.* Springer, Berlin, Heidelberg, 68–83.
[8] Meike Klettke, Uta Störl, and Stefanie Scherzinger. 2015. Schema Extraction and Structural Outlier Detection for JSON-based NoSQL Data Stores. In *Proc. BTW'15.* GI e.V., Bonn, 425–444.
[9] Mark Lukas Möller, Nicolas Berton, Meike Klettke, et al. 2019. jHound: Large-Scale Profiling of Open JSON Data. In *BTW'19 (LNI).* GI e.V, Bonn, 555–558.
[10] Diego Sevilla Ruiz, Severino Feliciano Morales, and Jesús García Molina. 2015. Inferring Versioned Schemas from NoSQL Databases and Its Applications. In *Conceptual Modeling.* Springer International Publishing, Cham, 467–480.

[3]https://stadt-zuerich.ch/geodaten/download/Spital?format=10009 [2020-09-21].

[4]https://sites.google.com/view/evolving-nosql/tools/darwin