WIRTSCHAFTS
UNIVERSITÄT
WIEN VIENNA
UNIVERSITY OF
ECONOMICS
AND BUSINESS

Master Thesis

A structured approach to Open Data monitoring

Florian Jauernig

Date of Birth: 01.03.1997
Student ID: 01613073

Subject Area: Digital Economy

Degree Program Code: UJ 066 960

Supervisor: Univ.Prof. Dr. Axel Polleres

Date of Submission: September 25, 2023

Department of Information Systems and Operations, Vienna University of Economics and Business, Welthandelsplatz 1, 1020 Vienna, Austria

DEPARTMENT FÜR INFORMATIONS-
VERARBEITUNG UND PROZESS-
MANAGEMENT DEPARTMENT
OF INFORMATION SYSTEMS AND
OPERATIONS

Contents

Abbreviations

AI Artificial Intelligence.

API Application Programming Interface.

CKAN Comprehensive Knowledge Archive Network.

CSV Comma-Separated Values.

DCAT Data Catalog Vocabulary.

DCAT-AP DCAT Application Profile.

HTML Hypertext Markup Language.

HTTP Hypertext Transfer Protocol.

HTTPS Hypertext Transfer Protocol Secure.

JSON JavaScript Object Notation.

ML Machine Learning.

ODArchiver Open Dataset Archiver.

ODPW Open Data Portal Watch.

OGD Open Government Data.

RDF Resource Description Framework.

URL Uniform Resource Locator.

List of Figures

List of Tables

Code Listings

Abstract

Open Data is freely available data published by governments or companies and while it can improve transparency for citizens and fuel innovation, its value is highly dependent on the quality of the datasets and associated metadata. Meanwhile, the Open Data landscape is evolving, with new data portals going online while others are discontinued or change their technical implementations. Building on prior research at WU, our new Data Portal Tracker allows automated discovery and validation of portals and extends the ODArchiver, a prior dataset crawling and archiving tool, to also index metadata. By providing detailed technical documentation, we want to enable future extensions, such as a metadata quality rating system. Our initial results show 2.7 times as many portals as the previous reference list used in many WU projects, making the output of our system comparable to more resource-intensive and less dynamic solutions found in literature.

1 Introduction

Open Data is data made available for unrestricted use [15], and is typically published by governments or companies along with respective metadata in **Open Data portals**, which are often accessible through application programming interfaces (APIs) of standard data catalog frameworks such as CKAN, Opendatasoft or Socrata [23, 26]. Some of the biggest drivers of the movement are **Open Government Data (OGD)** initiatives which are aimed at improving transparency [20] and stimulating the economy through innovation [1]. Increasingly, also companies and semi-public organisations recognize the value and innovation power of Open Data, however, mostly through collaborations with other firms as opposed to advertising and exchanging their data offerings through own data portals [36, 42].

Naturally, as time passes and more data gets published, two questions come into focus: how does the quality and availability of data differ and are there any significant changes over time? A notable example of efforts in the corresponding field of **Open Data monitoring** is the official portal for Open Data originating from countries in the European Union, **data.europa.eu**, which features a dashboard showing metadata quality scores and their historic evolution for all indexed portals [30]. Previously, **OpenDataMonitor**, a project co-founded by the European Union, provided a similar set of features, however, the platform seems to not be maintained anymore [31].

WU's institute of Data, Process and Knowledge Management has actively taken part in this field through the development of solutions for monitoring and archiving metadata and datasets from publicly available Open Data APIs, two important examples being the **Open Data Portal Watch (ODPW)**, a monitoring framework for assessing the quality and status of Open Data portal APIs [26] and the **Open Dataset Archiver (ODArchiver)**, a crawling and archiving tool that downloads the datasets offered by portals and checks for new versions in dynamic intervals [40].

However, since data.europa.eu is limited to data portals from the EU (many of which are by EU institutions), OpenDataMonitor is not functional anymore apart from displaying a list of portals, WU's ODPW is also no longer operational and other internal WU projects were never completed or have different goals, there is a gap in portal monitoring capabilities, both inside and outside WU, since there is no project or tool keeping track of data portals worldwide [30, 31]. Additionally, most monitoring solutions and lists of portals have so far been based on manual collection of relevant websites and only rate the metadata quality, but do not archive the actual datasets. Research indicates the feasibility of automated **portal discovery**,

albeit with heavy resource usage [7, 8], and the institute already has a tool capable of archiving datasets [40]. Thus, the system to be developed as part of this thesis should contribute by building an integrated infrastructure that can be used for semi-automated portal discovery and monitoring of data portals as well as archiving of datasets and metadata. Furthermore, it should lay the foundation for subsequent analyses, particularly the calculation of metadata quality scores in a future reproducibility study of the ODPW paper by Neumaier et al. [26].

1.1 Research Problem and Research Questions

We have identified the following research problems and gaps which directly lead to a set of research questions that we aim to tackle both conceptually and with a practically usable implementation.

The first issue concerns the discovery of data portals. For many Open Data related system developments and research works carried out in the institute, the list of portal APIs that was originally created for ODPW serves as the central input [21, 25]. The ODArchiver, while intended to handle arbitrary URLs, has also used the same list to build up its corpus of regularly crawled data [40]. Neumaier et al. briefly mention the various sources used for this list, but do not present any selection criteria or a repeatable and structured process for creating the list [26]. The reason for this is that in the beginning of the project, a list of available portals was mostly manually picked and collected through personal communications, with the initial goal to find as many such portals in a best effort approach. In the past years, many new portals have gone online, whereas others were discontinued or migrated to new data offerings, or changed their APIs. Consequently, multiple questions arise: How can we create the list of monitored portals in a sustainable and automated fashion? Where can we get the URLs of services that are built on a specific data catalog software? How can we check the ongoing availability of the APIs? How can we continuously update the list by adding new entries and marking or removing inactive ones? From these open issues, the **first research question** is derived.

Over time, the fragmented nature of the many different projects and undertakings has lead to an urgent to stabilize the existing infrastructure and tie the separate but related parts together in a meaningful way. The ODArchiver is the most promising tool in terms of capability, but its metadata handling must still be extended.

While both internal and external solutions for the continuous monitor-

ing and metadata assessment of data portals exist, most of the discussed platforms are currently not working in a stable and reliable way (OpenDataMonitor has remained online longer than ODPW's website but also already has major issues with broken functionality) and have stopped taking metadata snapshots years ago [31]. The only exception is data.europa.eu, but it is limited to selected European portals [30]. In 2020, the ODArchiver started crawling and storing datasets, however, it does not store the metadata provided by the data portals, only some automatically generated file-related metadata [40]. As a consequence, we are currently unable to conduct long-term analyses of the evolution in the field. The previously existing monitoring capabilities must be fully restored to help conduct further valuable research and enable a follow-up / reproducibility study that should answer the **second research question**.

Research question 1: *Can a comprehensive list of Open Data portal APIs be retrieved from the web and regularly updated in an automated way?*

Research question 2: *How can a scalable infrastructure for monitoring the evolution of Open Data portals, their datasets and metadata be built?*

Following from the research questions, the first goal of the thesis is to introduce a structured way of (semi-)automatically generating a list of portals that is much larger than the list created for ODPW. Additionally, we will use the new method to validate the ODPW list by checking the availability and functionality of portals on it. The second goal is to build a sustainable and scalable infrastructure that continuously collects and stores relevant metadata to enable the subsequent calculation of quality metrics. We will not conduct a full reproducibility study, but rather lay the foundation for other researchers at our institute to subsequently analyze the evolution of Open Data APIs in continuation of the ODPW efforts.

To solve the outlined research problems, our work will comprise both a thorough literature review and practical programming work. Based on the issue at hand and the associated research questions, we will research the relevant background and preliminaries regarding Open Data, Open Data portals, portal discovery, Open Data monitoring and related topics. Subsequently, the existing technology stack of the Open Data related infrastructure at the institute will be closely examined to derive current capabilities and shortcomings that will then inform functional goals and technical requirements, which are essential for the practical implementation. Results will be validated using suitable metrics and the system will be documented in detail and made available via a Git repository.

1.2 Thesis Structure

The remaining sections of this thesis are outlined below.

Chapter 2 introduces the main concepts and preliminaries of Open Data, presents important examples of existing solutions regarding portal lists and metadata quality assessments and highlights related literature in the covered fields.

Chapter 3 includes a detailed analysis of the existing infrastructure and previously developed systems at the institute, highlights the need for a new integrated system to restore and extend past capabilities and lists the derived goals and requirements and ways of evaluating the finished product.

Chapter 4 presents the Data Portal Tracker, the new system developed for this thesis, and shows how it connects to the existing infrastructure and how each of the components are implemented.

Chapter 5 showcases and evaluates the first results obtained by using the Data Portal Tracker, mentions the identified weaknesses of the system and suggests improvements as well as possible future research topics and applications for the Data Portal Tracker.

Chapter 6 closes the thesis with a short conclusion that summarizes the main contributions and gives an outlook into the future.

The **documentation** in the appendix serves as a comprehensive guide that will enable researchers at WU to understand, use, improve and extend all components of the Data Portal Tracker.

2 Background and Related Work

Before assessing the current state of the institute's infrastructure and for a better understanding of the issues at hand, it is important to get an overview of the relevant basic concepts and look at some key facts and the impact of Open Data in research and practice while discussing different approaches and recent regulatory changes.

2.1 Foundations of Open Data

Geiger and von Lucke define **Open Data** as follows:

> *"Open Data are all stored data which could be made accessible in a public interest without any restrictions for usage and distribution."* [15]

If this data is made available by public bodies, it is referred to as **Open Government Data (OGD)** [15]. In fact, governments around the world are a major contributor to Open Data as they publish data about the administrative resource use [20] as well as statistics about public bodies, citizens and businesses and make the data available via **Open Data portals**, for example on data.gv.at, the Austrian OGD portal run by Austria's Federal Ministry of Finance.

Prior research has shown the importance of Open Data portals in improving **transparency towards citizens** as part of Open Government initiatives and has identified the following essential aspects to guide a transparency-by-design approach when building a data portal: quality, accessibility, findability, understandability and usefulness of data as well as portal structure, community engagement and user support [20].

Companies and semi-public organisations have two main options when working with Open Data. Firstly, the **commercial use of Open Data** provided by outside organizations, such as governments and other corporations, to create or improve products and services [42] and secondly, the **sharing of internal company data** with other firms or the general public, for example to increase collaboration and innovation [36].

In 2000, a study published by the European Commission attempted to quantify OGD usage by businesses and estimated the value of public sector information at over 1 % of the GDP in the European Union, with a majority of the value being created through the use of geographical, economic, social and business-related data. More precisely, the economic value was estimated to be between EUR 28 billion and EUR 134 billion compared to a yearly investment of EUR 9.5 billion [12]. 13 years later, a survey conducted among

Swedish IT companies showed that the use of OGD was perceived as highly important and in some cases indispensable for the concept and core activities of the analyzed companies [19].

Data originating from companies, however, still makes up a smaller part of the Open Data ecosystem, with only some firms opting to advertise and exchange their data offerings though such portals and hoping to benefit from increased collaboration and innovation [36, 42]. Even among these companies, few seem to make data publicly available, but rather focus on **Open Innovation** efforts, which are collaborative research projects of a limited number of companies that help innovation and reduce costs and time for the participants compared to individual attempts [3].

In order for businesses to create value using data published by governments, they require personnel with specific skills, data availability, data and information capabilities as well as hardware and software resources [42], while the outcomes of Open Innovation have been shown to also be dependent on the extent of management support [3].

Recently, the potential economic benefits of Open Data have motivated the institutions of the European Union to step in and modify previously existing legislation on public sector information to help Open Data adoption. In June 2019, the European Parliament and Council adopted **Directive (EU) 2019/1024 on Open Data** which aims to increase the use of Open Data and to fuel innovation. It applies to public bodies and, with certain limitations, also to public companies and publicly funded research projects. One of the goals is to support Artificial Intelligence applications by limiting the costs of data provision, encouraging the use of APIs when making data available and mandating the publication of government data in areas like meteorology, geospatial mapping and mobility [1]. Regarding this obligation for OGD publishing, the directive requires the European Commission to compile a list of high-value datasets in a separate regulation, which was done in the Commission Implementing Regulation (EU) 2023/138 in December 2022. With a high level of detail, the regulation defines the thematic scope, structure and exact content of datasets that are considered to have substantial value for the economy and society at large and must therefore be made available free of charge in standard formats [2].

Open Data portals typically provide **datasets** consisting of a number of **resources** which are the actual data files that contain structured data (e.g. tables) or unstructured data (e.g. text documents). Accompanying **metadata** informs users on the content, structure, origin, and many other aspects of the dataset [26]. Typically, both a web interface and an API can be used to retrieve data from a portal.

Operators of Open Data portals often do not develop a custom solution from scratch to host their data, but rather rely on existing **portal software frameworks** (which we will later also refer to as data management systems or data catalog software). Popular choices include CKAN (Comprehensive Knowledge Archive Network), Socrata and Opendatasoft [26], which will be essential in subsequent chapters of the thesis. Our example from above, data.gv.at, is currently using CKAN, which is evident from the response to a CKAN-specific "status_show" API function call.

An important distinction to make when dealing with different portal software concerns the **terminology** and fundamental design choices regarding datasets. On portals that use Opendatasoft or Socrata, a dataset is often equivalent to a resource on CKAN portals in terms of content. CKAN allows multiple resources to be grouped together in one larger entity which is called a dataset or a package. These resources may be multiple versions of the same file in different formats or multiple files with different content (or a combination of both). In contrast, if multiple download URLs are available for a single dataset on an Opendatasoft or Socrata portal, this is because the same dataset is offered for download in different file formats. In this case, the content is typically the same regardless of which download URL is used, which is the key difference to CKAN.

In interviews with providers and users of Open Data, Janssen et al. identified deficiencies in availability and **standardization of metadata**, among others, as issues hindering Open Data adoption. Adding to the problem are the different interests of users one one side, for whom metadata directly impacts the findability and usability of datasets and some government officials on the other, who expressed the wish to publish data without any work-intensive metadata-related edits [16].

Naturally, with a rising number of data providers, different ways of expressing metadata have emerged. These so-called schemas differ in the presence and naming of various fields, which complicates finding, using and exchanging large quantities of datasets from different sources. One possible way of addressing this issue is **Data Catalog Vocabulary (DCAT)**, a recommendation by the World Wide Web Consortium (W3C) that offers a standardized metadata schema. Based on the needs of European data providers, the European Commission has created an extension to DCAT, the DCAT Application Profile (DCAT-AP) which is in use across many OGD portals in Europe, benefits the integration of portals in platforms like data.europa.eu and has in turn inspired changes to the standard, which were introduced in version 2 of DCAT [18, 26, 39]. Another standard vocabulary enabling unified metadata schemas is the **Schema.org dataset vocabulary** [27] which

is used in the Google Dataset Search alongside DCAT [4].

2.2 Monitoring Approaches and Common Metrics

When it comes to Open Data monitoring, two of the most important activities that existing monitoring efforts and services engage in are the following:

1. Creating and updating lists of data portals and possibly making them available on the web. There may be restrictions to certain regions (e.g. Europe) or areas (e.g. research data, government data). Regardless of the method used and even if the list is not published on the web, this **data portal discovery** is a requirement for rating the quality and tracking the evolution of portals.

2. Analyzing data portals, particularly by examining their datasets, resources and associated metadata and giving quality scores related to features like availability, discoverability or usability. When carrying out **data portal monitoring**, these ratings may be repeated regularly, enabling comparisons of current and historic results.

As mentioned above, to monitor portals and rate their metadata quality, their existence must first be known, therefore a major aspect in this context is the **data portal discovery**. Previous solutions such as ODPW have often relied on manually created lists. Today, multiple large lists of portals are available online, but the collection procedure is usually also manual. Examples include a list provided by the Open Knowledge Foundation that was manually curated by domain experts [14] and another one by Opendatasoft that was put together from multiple sources by hand and is described as "crowdsourced" [33]. On OpenDataMonitor, a platform mentioned earlier that used to provide metadata quality ratings, a portal list is still online that was compiled using a manual process, as the project's knowledge base includes a note that automatic discovery is not supported [32]. Services like re3data and Fairsharing, which are conceptually similar but focus more on research databases and repositories instead of data portals by governments or companies, also curate the data by hand and manually evaluate suggestions for new entries according to specific guidelines [28, 29]. One exception seems to be a list of websites using CKAN which was compiled by BuiltWith in what they claim to be an automated or semi-automated way using web crawling. However, the process is not transparent and the list is not freely available - rather, a subscription costing 295 USD per month is required to access the information [5].

The validation and automated discovery of data portals has also been investigated in academia, for example in a series of three research papers by Correa et al. In 2018, the authors combined and deduplicated data from various portal lists, including WU's ODPW, resulting in 3152 unique URLs. A subsequent check of the sites' API functionality showed a total of 356 working CKAN, Opendatasoft and Socrata portals [9].

A year later, an adapted version of the survey was repeated. This time, the portal list to be checked was sourced from the URL index of the Common Crawl archive from November 2018 containing more than 3 billion web pages and filtered by keeping only URLs containing the keyword "data" in multiple languages. 719 working CKAN, Opendatasoft and Socrata portals were found this way [8].

In the third study, Correa et al. repeated a similar approach based on Common Crawl data from April 2019 and performed a search for "open data" and all three software names in the HTML code of 2.5 billion web pages, processing around 200 TB of data. Almost 1 million potential portals with keyword matches were found and subsequently multiple requests were sent to them to look for APIs, resulting in 837 working CKAN, Opendatasoft and Socrata portals. They then trained an ML algorithm on labeled data consisting of the HTML source code of the validated data portals and of other websites. However, the testing datasets were unlabeled, therefore no metrics like accuracy could be calculated for the model [7].

A possible way to avoid the high utilization of resources in the studies above could be focused crawling, a concept proposed by Chakrabarti et al. in 1999, which moves away from the common and hardware-intensive approach of crawling the entire World Wide Web and subsequently searching the complete crawl using keywords. Instead, the authors document the advantages of a crawler based on an ML model pre-trained on relevant websites which classifies each visited site and only follows the links further if the current website is classified as related to the topic of interest [6].

An example for existing **data portal monitoring** offerings, marketed as the central portal for Open Data in the European Union, is **data.europa.eu**, which features a metadata quality dashboard. Since 2021, it has been rating portals using multiple attributes and has been providing portal rankings, aggregated results and the evolution of portal ratings [30]. The former European Data Portal, one of the predecessors of data.europa.eu, had originally been built on CKAN, but due to scaling issues related to the translation of metadata into RDF, a custom data management system called Piveau was developed for the platform [18]. Another example was the website **OpenDataMonitor**, which was co-founded by the European Union. It featured

quality and quantity metrics and showed their evolution over certain time periods, both for individual data portals and aggregated per country. When most parts of the website could still be accessed without issues, the data shown had not been updated in multiple years and by now, the metadata ratings are not available anymore [31]. At WU, **Open Data Portal Watch (ODPW)** was a monitoring framework for assessing the quality and status of Open Data portal APIs by checking the availability of the API as well as downloading the metadata of resources and using it to calculate quality scores. ODPW solved the issue of differences between metadata schemas of the data catalog software providers CKAN, Socrata and Opendatasoft by mapping the schemas to the Data Catalog Vocabulary (DCAT), a standard metadata vocabulary defined in RDF [26, 39]. In a related project, the metadata was further mapped to the Schema.org dataset vocabulary [27].

Many of the **quality metrics** commonly used in portal monitoring solutions share similarities with the criteria for a transparency-by-design approach when building a data portal that we summarized earlier [20]. To calculate scores for ODPW, Neumaier et al. incorporated the following aspects [26]:

- Existence (of metadata)
- Conformance (to formats)
- Retrievability (of data)
- Accuracy (of metadata)
- Open Data (openness of data)

Wentzel et al. have developed a system for rating metadata that is mostly based on previous work like the FAIR principles and Five-Star Linked Open Data and has been deployed on the data.europa.eu platform. Their metrics are organized in the following categories [41]:

- Findability
- Accessibility
- Interoperability
- Reusability
- Contextuality

2.3 Related Work

In this section, we present selected projects and research papers that are either related to previously cited literature or cover topics that are out of scope of this thesis but are still relevant and could inspire future extensions of our work.

ADEQUATe, a project that ran from 2015 to 2018, dealt with the question which quality aspects and information contained in metadata are important to users and lead to the creation of a platform for assessing and improving the quality of datasets from selected Open Data portals with the help of user contributions. It included a Data Monitor module for retrieving metadata that was based on the ODPW framework [24].

In a recent paper, Kirstein et al. tackle an issue that we might also encounter in the development of our system: the need for unique, cross-platform **persistent identifiers** for datasets. The authors propose replacing the Internationalized Resource Identifer (IRI) recommended by DCAT with a decentralized solution based on a distributed ledger architecture consisting of nodes that assign and resolve unique identifiers [17].

Benjelloun et al. provide insights into **Google Dataset Search**, a search engine for datasets, and describe how it indexes all datasets located on websites that can be crawled by the Google Crawler and offer schema.org or DCAT metadata. As of 2020, the corpus contained 28 million datasets, 90% of which could be classified as Open Data based on the metadata, and showed high rates of activity regarding deletions and new additions. A mere 44% of metadata included a dataset download URL, which might lead to challenges in our project, even if the portals that we will analyze will typically not be research data repositories like the largest sources of Google Dataset Search.

3 Current State and Goals

We will now analyze the existing infrastructure and previous projects related to Open Data monitoring at WU's *Institute for Data, Process and Knowledge Management* based on literature, database queries and personal communication and point out their strengths and weaknesses. Once we have evaluated the current situation, we will lay the groundwork for the development of a new integrated solution by defining functional goals (what should the system be able do?) and technical requirements (how should the system do it?) which will guide our evaluation of the finished product.

3.1 Current Infrastructure

Figure 1 gives an overview of the current Open Data infrastructure and how the systems and data relate to each other. From 2016 to 2019, ODPW crawled and rated **metadata** and made the aggregated results for each portal available on a website. ODArchiver has been downloading, storing and versioning **datasets** since 2020 and features an API and a website. In 2021, Portal Watch API attempted to replace ODPW with the help of the ODArchiver, but the project was never finished. All of these systems use the same list of portals that was originally created as part of the ODPW project.

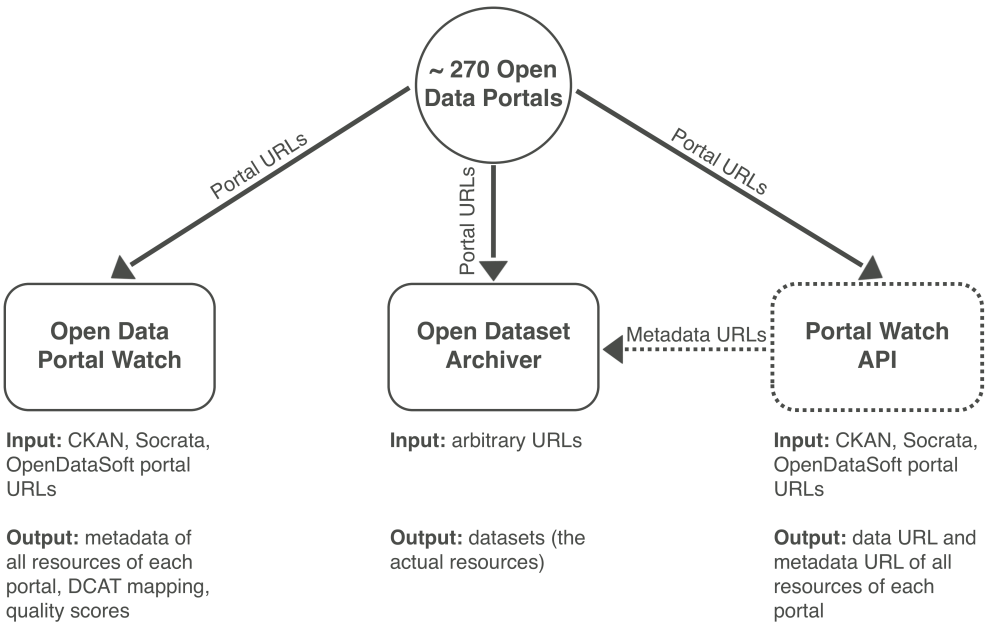

Figure 1: Relevant parts of the current infrastructure.

Open Data Portal Watch (ODPW) is a service that periodically took snapshots of the metadata of more than 200 CKAN, Socrata and Opendatasoft portals, checked the availability of their API endpoints and made results for each portal available on the project's website. It was designed by Neumaier et al. to calculate metadata quality scores based on a number of attributes related to the categories *existence*, *conformance*, *retrievability*, *accuracy* and *Open Data* [26]. Different metadata schemas of CKAN, Socrata and Opendatasoft were mapped to the standard RDF metadata vocabulary of DCAT [39]. Database queries show that snapshots were taken from June 6th, 2016 to August 19th, 2019, with the number of snapshots varying depending on the database table. Table *portalsnapshot*, which contains basic properties like the number of datasets and resources for each portal, shows 162 snapshots. Meanwhile tables *datasets* and *resourcesinfo*, which provide more information about the individual datasets and resources of each portal, include 133 and 85 snapshots, respectively. Figure 2, which outlines the architecture of ODPW, shows that the system harvested metadata from the selected portals and sent head requests for the resources, but does not give any information on how ODPW handled the portal discovery. This is because the list of portals used by the system was manually created and updated while the project was running. The problem we are facing regarding ODPW is that the system is no longer fully operational, as it has stopped taking snapshots 4 years ago and stability issues leave its website offline more often than not. Unfortunately, since the project was programmed in Python 2 and has some limitations regarding scalability and efficiency, fixing and modernizing ODPW would entail a full rewrite. Thus, to resume Open Data monitoring, a replacement for ODPW is needed.

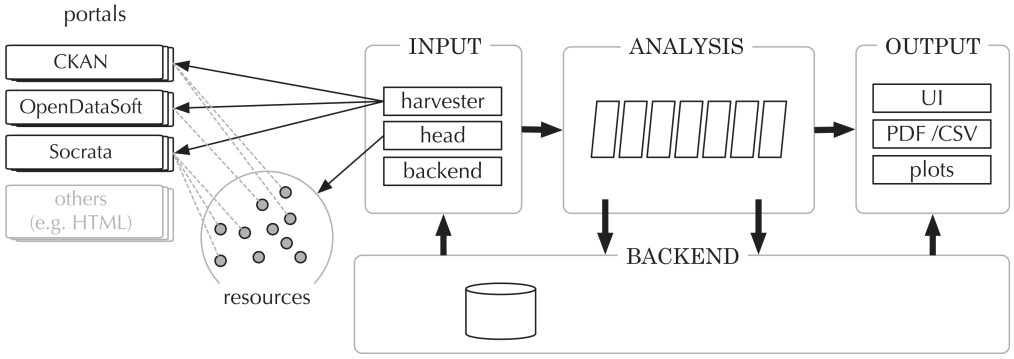

Figure 2: Architecture of ODPW [26].

Open Dataset Archiver (ODArchiver) is a web crawling and data archiving tool that downloads, stores and versions the **data** / **resources** of

datasets, is compatible with many different file formats and can take arbitrary input URLs. As figure 3 shows, it uses NGINX for the load balancer (and reverse proxy) and Node.js for carrying out tasks in the backend while MongoDB serves as the database. Due to its use of a Kubernetes cluster with multiple nodes, the ODArchiver is also highly scalable. Since April 6th, 2020, the ODArchiver has been regularly crawling an initial corpus of datasets based on ODPW's list of portals. Currently, the ODArchver only stores a very limited amount of metadata that is focused more on the crawling process whereas the detailed metadata provided by portals is not downloaded [40]. With its strong scalability and file compatibility, the ODArchiver could be used to crawl not just datasets but also metadata by adding the dataset URLs and metadata URLs via its API and creating a mapping system between datasets and their related metadata. In fact, this was the idea guiding the Portal Watch API project, which is described next.

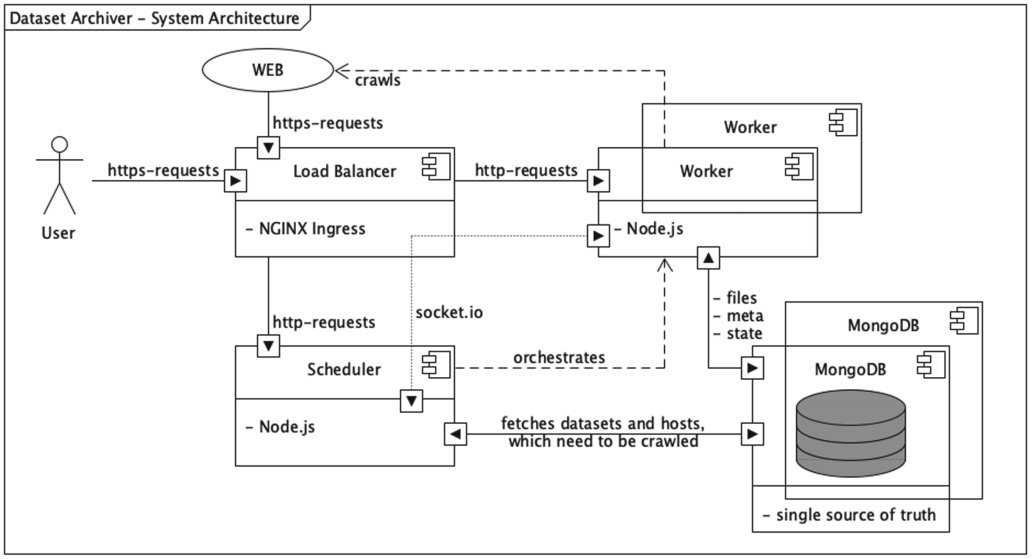

Figure 3: Architecture of ODArchiver [40].

Portal Watch API is an unfinished tool that was intended to accept a list of CKAN, Socrata, and Opendatasoft portal URLs as input and retrieve both the download URL and the metadata URL of each individual resource of every portal before adding them to the ODArchiver. Contrary to its name, but just like the ODArchiver, the important parts of Portal Watch API's code are completely independent from the ODPW system. Another similarity to the ODArchiver is that it is designed to get its input from ODPW's list of portals, however, in this case, this can be easily changed [10]. Portal Watch

API was a first attempt at restoring past capabilities by replacing ODPW, but was discontinued in an early stage. Some of the ideas and logic contained in its Python scripts will serve as the basis for parts of the code created in the implementation phase of this thesis.

Datamonitor was another data collection effort for which a database is still available that holds crawled data mainly from 2016 to 2019, with some tables indicating that crawling started as early as 2014 and 2015. However, the relevant table *crawldata* seems to only contain aggregated counts of domains and file formats of datasets. Additionally, some CSV files were downloaded from 2014 to 2019, but the data is only very limited in most years at the start and end of the range. Unfortunately, the database does not seem to contain datasets or metadata that would be relevant to us in terms of quality or quantity.

3.2 Functional Goals

Based on the identified research problems, the research questions and the current state of the infrastructure, there are three main functionalities the new system should be able to provide:

1. It should feature a **dynamic / semi-automated discovery** of available data portal APIs using a structured and repeatable method to create, validate and update the portal list. There should be an option to add portals manually to handle special cases.

2. Partly consolidated from previously available tools and partly newly developed, the resulting architecture should fulfill the tasks of **automated crawling and portal monitoring**. In addition to integrating the archiving of datasets and metadata, it should also create a mapping between them to facilitate working with the data.

3. To **prepare reproducibility studies** of the original study by Neumaier et al. [26], the new system should be deployed to access the portals from ODPW's list and evaluate the availability of the portals as well as one example metric from the study. Additionally, there should be a workflow for accessing the metadata as this is relevant for future work.

In table 1, the functional goals derived from these desired functionalities are presented alongside a code that will enable us to easily refer to specific goals and more precisely evaluate whether the implemented system meets the requirements.

Table 1: Functional goals of the new system.

Code	Description
F.1	**Automate portal discovery and validation**
F1.1	Find portal URLs on the web in an automated way
F1.2	Validate the used portal software
F1.3	Allow manual additions of portals
F.2	**Restore and extend Open Data monitoring capabilities**
F.2.1	Integrate dataset and metadata archiving
F.2.2	Create a mapping between datasets and metadata
F.3	**Prepare ODPW reproducibility studies**
F3.1	Validate ODPW's portal list
F3.2	Show workflow for accessing metadata

3.3 Technical Requirements

After having determined what our new system should achieve, we will now also define how exactly the solution should be implemented and what the requirements regarding performance, maintenance and documentation are:

1. Internal and external **resource use**, particularly in terms of storage and HTTP requests, should be kept as low as possible. Due to the quantitative nature of this requirement, we will be able to try out multiple options and compare their results and effects before making decisions on implementation details.

2. The system should take advantage of the scalable architecture of the **ODArchiver** and add datasets and metadata via its API while adding the mapping between them into a new collection in the existing MongoDB. Also, there should be backwards compatibility with ODArchiver's corpus by ensuring that the system can add any missing part (dataset, metadata or dataset/metadata mapping) individually so that existing datasets already indexed by the ODArchiver can be enriched.

3. To simplify future maintenance and extensions, the system should be **well documented** and written in Python 3 which many institute staff members and informatics students are familiar with.

Just like for the functional goals, we assigned codes to the technical requirements in table 2 shown below, which will be used in subsequent chapters.

Table 2: Technical requirements of the new system.

Code	**Description**
T.1	**Manage resource use**
T1.1	Keep hardware use low during portal discovery
T1.2	Minimize HTTP requests during portal discovery and validation
T1.3	Balance URL deduplication and information loss
T.2	**Use and extend ODArchiver's infrastructure**
T.2.1	Use ODArchiver's API for adding datasets and metadata
T.2.2	Add a new mapping collection to ODArchiver's MongoDB
T.2.3	Ensure compatibility with old datasets indexed by ODArchiver
T.3	**Create a sustainable and well documented architecture**
T3.1	Write code in Python 3
T3.2	Use comments, docstrings, README files and Jupyter notebooks
T3.3	Provide a detailed guide on using and extending the system

4 Implementation

In this chapter, we present the Data Portal Tracker, which attempts to achieve goals F.1 to F.3 in the given order while meeting requirements T.1 to T.3. We will highlight its main aspects, starting with the portal discovery and then moving on to the portal crawling and the preparation of reproducibility studies. Before going into implementation details, however, we will give an overview of the system.

4.1 System Overview

Figure 4 shows the major elements of the Data Portal Tracker, our new system for discovering, validating and monitoring Open Data portals and regularly crawling and archiving their datasets and metadata. The components for search engine crawling and portal validation include shortened and simplified code from the tool *Crawley* by Dobriy [11], while the *ODArchiver* by Weber et al. [40] is used as for downloading and permanently storing datasets and metadata. First drafts of the portal crawling and ODArchiver connection code were inspired by the unfinished *Portal Watch API* [10].

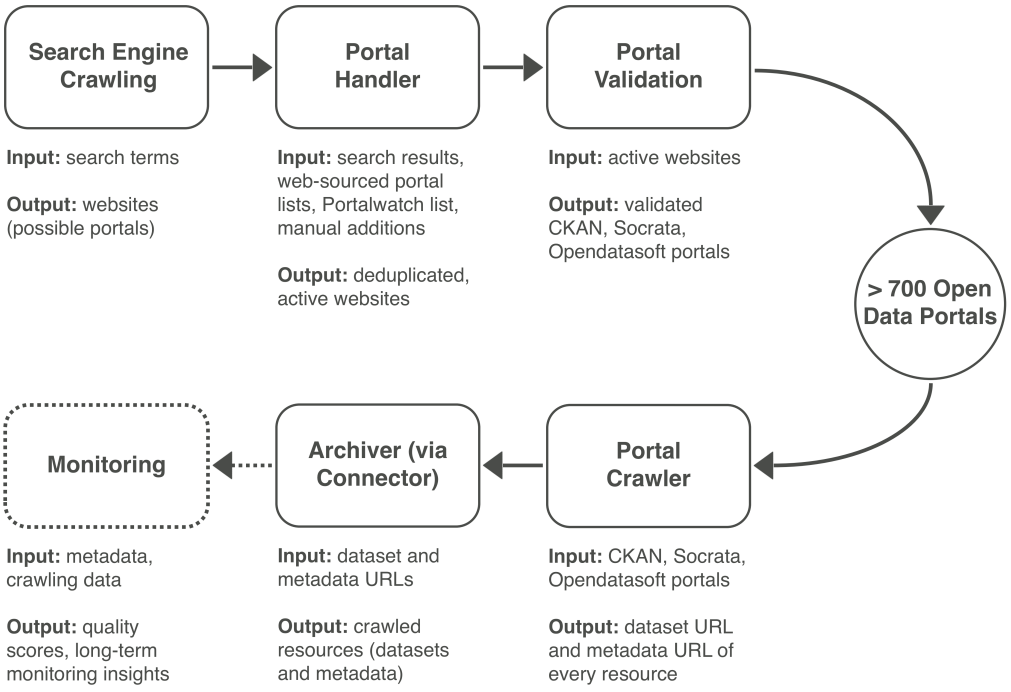

Figure 4: Data Portal Tracker components.

Data Portal Tracker's code is available via a Git repository[1]. The project's root directory contains a README file, a text file listing the requirements and two directories crawley-lite and data_portal_tracker which are intended to keep external and internal code somewhat separated despite a tighter integration than typical of a dependency. For development, experiments and data exploration, we made heavy use of Jupyter notebooks in the data_portal_tracker directory. In the subsequent sections, we describe the implementation process and functionality of the system, while the README file and the documentation in the appendix focus on usage and deployment aspects.

4.2 Portal Discovery and Validation

First, we will describe the portal discovery and validation pipeline, which consists of the components "Search Engine Crawling", "Portal Handler" and "Portal Validation" visible in figure 4 and covers the activities of finding potential portal URLs, adding them to a list, validating the portals in a multi-step process, extracting data that is relevant for the subsequent components and analyzing the results. The relevant code can be found in the crawley-lite.py script in the directory of the same name as well as the portal_handler.(ipynb|py) script in the data_portal_tracker directory.

Starting with **search engine crawling**, we use the tool *Crawley* created for the institute by Daniil Dobriy [11] to perform search engine queries via the service SerpAPI, but have reduced the code to the core functionality required for the Data Portal Tracker. Compared to other semi-automated solutions for portal list creation researched previously [7, 8], we are exploring a different approach by using *search engine portal discovery*, which aims to be similarly comprehensive while being less resource-intensive and easier to keep up to date. To collect Open Data portals dynamically and on the fly, our approach takes some inspiration from focused crawling [6] with regards to reducing the hardware requirements, but also differs in a few major ways. Most importantly, we are still querying comprehensive web crawls for relevant URLs, but not by downloading and searching through immense amounts of data like the archives of the Common Crawl. Instead, we are making use of the vast resources of large search engine operators like Google and are simply performing search engine queries via available APIs, then store the results and extract the URLs to be validated at a later stage. This approach requires a fraction of the resources of the previously published semi-automated discovery solutions [7, 8]. Alternatively, we could change the solution to resem-

[1] https://git.ai.wu.ac.at/h1613073/data_portal_tracker

ble focused crawling more closely by reimplementing recursive link crawling which is intentionally left out in our adapted and simplified version of Crawley. Recursive link crawling is possible in the original tool, but was not tested in depth as the results were already sufficient without using it, so it was one of the parts of the code to be removed for simplicity. Another reason for this and other edits was to improve future code maintenance seeing as Crawley is integrated differently than a normal dependency. However, if desired, this capability can be restored by copying any conditional sections in the original code asking for the command line argument --links to be specified, like "if args.links:", back to the crawley-lite/crawley-lite.py script [11]. In case of reimplementing recursive crawling, the results of the search engine queries could serve as a starting point, providing an initial list of sites whose links can then be crawled.

A major change compared to the standalone *Crawley* tool is the way search results are handled. In the original version, the URLs of all search results are crawled, with the full webpages being stored in the resultsHTML subdirectory. This means that for previously collected search results, pages do not have to be re-requested in a subsequent execution of the program. While this approach can be beneficial for certain use cases, it does not align with our system which is designed to rerun the pipeline completely from scratch for every creation of a new, updated portal list. By not requesting the webpages until later in the process when they are needed and not permanently storing them, we can mitigate this and reduce storage use while being able to seamlessly integrate the code into the portal validation. Thus, our adapted version does not have a resultsHTML folder, only a results folder in which the search result JSON files, which are later accessed by the portal handler to extract the URLs, are stored.

Next up is the portal_handler.(ipynb|py) script, which includes not just the "Portal Handler" component, but also the "Portal Validation" component from figure 4. It carries out the **portal list creation and validation** and currently features 4 data sources, taking input from the search engine results, from portal lists on the web, from ODPW's list of portals and from manual additions. The code deduplicates the domains, performs requests to all websites to check their activity status and whether HTTPS or HTTP is available and validates the remaining sites in two steps: First, it looks for HTML elements ("markers") that can usually be found on Opendatasoft, CKAN and Socrata sites. Then, it verifies that the API is actually working by requesting the specific endpoint and validating the response. Each time a list is created and validated, the most important statistics can be calculated and saved, allowing to monitor the evolution of the list over time.

During the deduplication process, URLs of possible portals taken from the different sources are shortened to their base URL by removing the path (/...). If this was not done, large numbers of duplicates would remain in the URL list, for example different subpages of the same website and sometimes even URLs of individual datasets. Our assumption here is that in most cases, the API can be reached by appending the appropriate API URL string (e.g. /api/explore/...) to the base URL. For deviating portals, the API endpoint can be researched and added by hand using a dedicated function. Such cases might be found by looking for portals which passed the first validation step, but not the second, meaning that portal software HTML markers were detected but no working API could be found using the base URL and the API string. In the CSV file created by the validation function, such portals have a non-null value for the "suspected_api" field other than "Unknown" while the "api_working" field has the value "False".

Apart from removing duplicate entries of the same portal, reducing the URL to its base URL also serves other purposes: completely erasing any erroneous characters that might occur at the end of a URL and avoiding the execution of denial-of-service attacks which, in extreme cases, might otherwise happen when checking the sites' activity and used data catalog software due to a multitude of requests.

In addition to removing the path from each URL, the protocol prefix is also truncated for better deduplication as some sites appear twice in the initial list, once with HTTP and once with HTTPS. Subsequently, when checking the activity status, we are adding a protocol prefix to URLs of active sites by testing whether HTTPS is available and falling back to HTTP if necessary. Our initial results validate this approach and show that even in 2023, we cannot simply assume that HTTPS is universally adopted. 702 out of 4380 active sites (16%) and 46 out of 705 working portals (6.5%) exclusively use HTTP, so only requesting websites via HTTPS would lead to information loss. Similarly, we tried removing the "www." prefix, assuming it was not required for sites to function anymore. However, a few portals are still requiring it, e.g. Bahrain and Corsica, so we decided to wait until the final list to remove any duplicate URLs with and without "www".

As mentioned above, sometimes the API endpoint of a portal is on a subpage, e.g. data.wu.ac.at/portal or data.gv.at/katalog. Since the URLs in the portal list are not always equivalent to the endpoint and the path is cut off completely in the deduplication step anyway, we are addressing this issue by manually adding such non-standard endpoints of selected portals with APIs that are known to be working. So far, the function `add_api_endpoints` has been implemented and integrated into the portal handler, but only the two examples from above have been added in this way. Extending the portal

list with more such cases could be a valuable contribution to improve and lengthen the portal list even further, but is out of scope of this thesis and thus listed in the future work section.

Certain portals pose challenges by combining two issues: not having any markers indicating the used software in their HTML code and exposing their API via an endpoint on a subpage. One example is data.overheid.nl/data, the Open Data portal of the Netherlands. Even in the unlikely case that the endpoint URL was present in some of the sources and thus part of the initial list, it would not persist until the final list for two reasons: In the deduplication step, the URL is reduced to its base URL and in the validation step, the API functionality of a site is only checked if any validation markers were found. To ensure that such portals still get included, the function mentioned above enables adding such API endpoints to the list after the deduplication step and marks them with a label that tells the validation function to skip searching for markers and to directly move on and check if the API works.

In general, when validation markers were found on a website but there is no working API (at least not reachable from the base URL), there are two possible explanations: either the marker-based validation produced a false positive as the site in question is not actually a data portal using the respective portal software or the second part of the validation that checks the API functionality incorrectly gave a negative result for a valid portal with a working API. Given the chosen approach of only testing if the API works on sites for which the validation marker search has been successful, these cases are worth investigating. For future improvements in this area, we suggest starting out with the basic code in the section "Checking false positives of marker validation" of the experiments notebook.

Finally, for a better understanding of the portal validation's output files, here are the specifics regarding retrieved and stored **version numbers** of working portal APIs:

- For **CKAN** portals, we store a single version number which is that of CKAN itself, not the number of CKAN's Action API which has only one available version [13].
- For **Opendatasoft** portals, we store an array of all supported version numbers among the Search API v1 and Explore API v2.0 / v2.1 [34, 35].
- For **Socrata** portals, the version number is that of the Socrata Metadata API which only has one version, not the number of the SODA API [37, 38].

4.3 Dataset and Metadata Crawling

Turning to the lower half of figure 4, we will now cover the components "Portal Crawler" and "Archiver (via Connector)" which are implemented in the portal_crawler.(ipynb|py) and archiver_connector.(ipynb|py) scripts in the data_portal_tracker directory. When executing one of the four available crawling functions, each for a specific portal software, the function takes the extracted portal information from the previous validation step, loops through all datasets or resources and adds them to the ODArchiver along with their metadata and a dataset/metadata mapping. Portal Watch API, an earlier effort to resume portal monitoring [10], was taken as a template for the mentioned scripts, but was almost completely rewritten and heavily extended - in effect, our final implementation only shares some ideas and logic with the original.

Our **portal crawler** works through the corpus of every data portal and retrieves three specific URLs for each dataset or resource that will be used by the ODArchiver later on:

- The **dataset URL** is a URL for downloading a dataset, but our definition of *dataset* depends on the portal software. For CKAN, it refers to a resource and we crawl all resources of a dataset/package separately, while for Opendatasoft and Socrata, it refers to one specific export option of a dataset, typically CSV, so we only crawl one file per dataset. For details on platform-specific terminology, see the chapter on foundations of Open Data. Previously, the ODArchiver often indexed multiple formats per dataset, as the URLs were taken from ODPW project using its SPARQL endpoint[2] [40].
- The **metadata URL** is a URL which allows fetching the metadata related to one specific dataset. Since CKAN provides metadata for each dataset, but not for each resource of a dataset, we are always crawling the URL of a *dataset's* metadata, regardless of the portal software.
- The **source URL** leads to a web page which is targeted at human users accessing it manually via a web browser and which presents information about a dataset and its download URL(s).

For crawling **Opendatasoft** portals, we support both the Search API v1 (deprecated) [35] and Explore API v2.1 [34]. Due to the large differences

[2]https://data.wu.ac.at/portalwatch/sparql

between these APIs, we developed two separate crawling functions. In either of them, we collect exactly one dataset URL, metadata URL and source URL per dataset, assuming that every dataset is available in CSV. To check the validity of this assumption, we carried out the following experiment:

Since Opendatasoft is specifically designed for structured datasets only[3], it could be hypothesized that every dataset on every Opendatasoft portal might be available in the CSV format. If true, this would allow a simplification of the Opendatasoft crawling function(s) and would reduce the required crawling time and number of HTTP requests, as the availability of the CSV file format would not have to be checked in every single instance.

Technically, the hypothesis would have to be the other way around: "Not every Opendatasoft portal offers every dataset in CSV". We could show that there is not sufficient evidence to prove this hypothesis if the number of datasets was the same as the number of supported datasets for every single portal which was successfully checked. While this would not prove that every Opendatasoft portal offers every dataset in CSV, it would should show that the opposite cannot be proven and would be a very strong indicator that for the purposes of simplification, the existance of the CSV format can be assumed without major information loss.

To test this, the crawl_opendatasoft_v2 function from the portal_crawler script checked every dataset on every Opendatasoft portal for the existence of a CSV file export option. In the portal statistics created by the function, we could see that out of 355 total Opendatasoft portals, the only ones for which the two relevant numbers do not match are 7 portals which could not be reached and 2 portals which offer so many datasets that the daily API call rate was exceeded by the function. Thus, it can be concluded that currently, every dataset on every Opendatasoft portal is available in CSV format.

Based on these results, the crawl_opendatasoft_v2 function was designed to always assume the availability of CSV. Due to this simplification compared to the experimental code, the number of requests per execution of the crawling function is reduced by 1 request per dataset (around 91000 as of August 2023), each taking about 0.5 to 1.5 seconds depending on whether a 1 second delay is used to lighten the load on the used APIs. This reduction also prevents the issue of reaching the API call limit of 5000 requests per portal per day which was encountered during the experiment (see figure 5) and would make it much more complicated to regularly check every dataset on Opendatasoft portals with more than 5000 datasets.

When using version 1 of the Opendatasoft Search API, assuming CSV availability is not an option that can be tested, but rather an inevitable

[3]https://help.opendatasoft.com/faq-glossary/en/faq_index.html

necessity since the documentation does not mention any method of retrieving the available file formats. On a side note, the filename is not preserved when downloading a dataset via Search API v1, so the downloaded CSV file is only named "download" and does not have a file extension.


```
{
  "error": "You have exceeded the requests limit for anonymous users.",
  "errorcode": 10005,
  "reset_time": "2023-09-04T00:00:00Z",
  "call_limit": 5000,
  "limit_time_unit": "day"
}
```


Figure 5: Opendatasoft API call limit.

For **Socrata**, monitoring the used API versions and adapting the code to ensure compatibility is less of a concern than for Opendatasoft because the Socrata Metadata API seems to only have one version [37] and while the SODA API, a closely related complementary API which allows advanced operations with datasets via the "/resource/" endpoint, has had multiple versions, the release intervals are extremely long, with the latest release (v2.1) dating back to 2015 [38]. Only datasets which are available in a tabular format or a file format like PDF, KML or ZIP are crawled, as we could not find consistent methods to build the dataset URL for other file formats. More specifically, there are multiple ways of obtaining access to a dataset. The first is by using the "/resource/" endpoint of the SODA API, which is designed to display the response directly on the site for browser users, and adding a file extension like .csv or .json to the URL. However, this only works for tabular data and so the chosen approach for the crawling script is to check the asset type and depending if it is a (tabular) dataset or a file (typically PDF, KML or ZIP), to build the dataset URL based on one of two different ways of downloading datasets. This means that only datasets which are available using these two methods are crawled and subsequently indexed - in the statistics file exported by the `crawl_socrata` function, these are referred to as "supported datasets" and counted alongside the total number of datasets, enabling us to evaluate the benefit of any extensions to be made to the script in the future. Currently unsupported types include charts, Data Lens pages, filtered views and maps - in section A.4 of the documentation, we discuss which types can likely be downloaded just like the "dataset" type, but still need to be tested.

Our **CKAN** crawling function has been tested with numerous CKAN versions ranging from v2.0 to v2.10 and works for all of them, possibly because

CKAN's Action API only has one version. When crawling CKAN portals, we are looping through the datasets using the "package_search" method and are only looking for resources, disregarding any datasets that do not have resources. The metadata URL saved in the process is that of the dataset's metadata, not the resource's (very limited) metadata, since the former includes the latter and additionally contains all of the important descriptive information about the resource. Since front-end webpages with information are only available for datasets, not for resources, the source URL is the portal's API base url plus the path "/dataset/" and the ID of the dataset or package. Finally, due to the differences in definition described above, we take the URL of the resource as the dataset URL.

To add crawled datasets and metadata to the **ODArchiver**, all of our crawling scripts call the handle_dataset function of our ODArchiver connector which sends requests to the ODArchiver's API and queries the new mapping collection in its MongoDB. First, the function checks if the ODArchiver is indexing the dataset and metadata already, adding one or both of them if necessary before getting the IDs assigned by the ODArchiver. Using the IDs, the existence of the dataset/metadata mapping is checked and in case it is missing, a new mapping is created.

When adding datasets or metadata to the ODArchiver, the relevant API method allows the file format to be optionally specified by passing it as an argument. However, there are no real benefits to passing this parameter, so it is intentionally omitted in the function. Also, the ODArchiver can already determine the file type and additional characteristics of tabular data by analyzing the file without relying on the file extension [40].

URL encoding is an important step when working with the ODArchiver API's /get/dataset/url method. Passing unencoded URLs to the method will not deliver consistent and reliable results as the lookup will fail and return an empty result for some datasets that are already indexed by the ODArchiver and for which MongoDB queries targeting "url.href" work without any issues. Subsequently, this makes the handle_dataset function think that the dataset is not indexed yet, and thus it tries to add it, which in turn does not work because the dataset is already in the ODArchiver.

Exact **duplicates** of URLs are recognized by the ODArchiver and so datasets are not indexed again when trying to add them repeatedly via the API. Additionally, our handle_dataset function always checks the existence of any dataset in the ODArchiver first. However, as there is no file-based deduplication, any cases where the same dataset in the same format can be downloaded via multiple different data repositories and thus has multiple

download URLs are not covered and could lead to duplicates.

In the process of finding out if a dataset is already included by the ODArchiver, the dataset URL serves as the identifier. However, when using the Opendatasoft Search API v1, the URL linking to the same file can be built in different ways, by adding or leaving out parameters like "format" and "csv_separator" or switching their order. To avoid duplicates, we need to create the dataset URLs in exactly the same way as when they were originally added to the ODArchiver. Therefore, we manually queried ODArchiver's MongoDB to check dataset URLs that were indexed from Opendatasoft portals using Search API v1, which showed that a majority of them used the format `api/records/1.0/download?dataset={ID}&format={FILE_FORMAT}`, so we used this exact URL format in our `crawl_opendatasoft_v1` function.

Since the introduction of the ODArchiver, Opendatasoft's Search API v1 has already been deprecated and we recommend using our function for the Explore API v2.1 instead. In version 2 and 2.1 of the Opendatasoft Explore API, the issue is mitigated because every export URL of a dataset always has the same URL structure, so there can't be multiple URLs pointing to the same dataset in the same format. However, a dataset might still be available in multiple file formats. For CKAN dataset URLs, there are no such issues since for every resource there is a single download URL, which is included in the respective metadata field. On Socrata portals, a dataset can also have multiple formats and the structure of a download URL depends on the file type, but we have not encountered more than one way of building the dataset URL for the same file format.

A more sustainable way around the issue, which would require changes to the ODArchiver, would be the addition of an API method that makes it possible to search for datasets by their source URL. The same could be achieved by using a MongoDB query that checks the source, however both variations of this approach have two major shortcomings: they rely on correct and complete source data being present for existing ODArchiver data and on the source URL remaining constant over time. Given the lack of a universal persistent identifier across the portals of interest, another possible solution would be the modification of the ODArchiver so that it saves the platform-specific identifier to a newly added metadata field. All in all, the issue highlights the **need for persistent identifiers** as means of ensuring the traceability of datasets and enabling long-term monitoring efforts. A possible solution proposed by Kirstein et al. [17] was mentioned earlier already.

Our **dataset/metadata mapping** implementation allows more than one dataset to be mapped to the same metadata in the new "datasets.mappings" collection of the ODArchiver's MongoDB, which is important since multiple

resources or multiple file formats of the same dataset all share the same metadata on CKAN, Opendatasoft and Socrata portals.

One of the additional requirements was to design a solution that considers changes of metadata and enables the extraction of insights like "this metadata described this dataset from A until B". Since the ODArchiver already has the capability to crawl indexed datasets periodically and keep track of versions, the decision to treat the detailed metadata as datasets leads to a simple and elegant solution, as the versioning features are therefore also extended to the metadata. Consequently, the database can be queried to retrieve and combine versioning information for any dataset and the metadata that describes it. By querying the collection "datasets.files" and specifying the "_id" value taken from the "datasets" collection as the "filename", it is possible to display information, including the upload timestamp, about all versions of a dataset, whether it is an actual dataset or metadata. The discussed requirement of analyzing the temporal component of each dataset-metadata relationship can thus be fulfilled by getting pairs of IDs from the collection that maps datasets to metadata and conducting queries in the described way. However, in an attempt to achieve easier access and to reduce execution times further, materializing the metadata mapping and versions at least partially could be useful.

In this context, it is also worth considering that the dataset or metadata URL is used as an identifier by the ODArchiver and therefore, if one of the two URLs changes, the dataset or metadata will not just be indexed again, but also a new mapping will be added to the collection if the handle_dataset function is called on the latest dataset and metadata URLs.

4.4 Reproducibility Study Preparation

While a full reproducibility study of ODPW is out of scope of this thesis, we will prepare future work in this direction by analyzing requirements and describing the necessary steps to extract metadata from the ODArchiver and map the different schemas to DCAT to simplify the metadata rating process. Additionally, we will use the Data Portal Tracker to validate the original ODPW list of portals and compare our results to those of a study from 2018.

Our system effectively extends the ODArchiver to index not just datasets, but also metadata. In the future, to fully restore the features of ODPW regarding **quality metrics**, metadata indexed by the ODArchiver must be accessed and analyzed. This brings two major requirements with it:

- The ODArchiver must be able to **download and store metadata in**

any form.

- It must be feasible to **extract metadata** indexed by the ODArchiver.

Addressing the first point, proof that the ODArchiver can handle arbitrary URLs, not just download URLs, and by extension metadata regardless of whether it is in JSON format or embedded into HTML is provided by previously indexed webpages[4].

With regards to the second requirement, to extract metadata, the desired version of the metadata needs to be identified, then the file belonging to that version must be found in the MongoDB collection "datasets.files". Afterwards, all chunked binaries from the collection "datasets.chunks" that make up the file must be rejoined and decoded using Base64. If the metadata is not directly present in JSON format, but is contained within HTML code, the relevant data then has to be extracted in a further step.

DCAT	CKAN	Socrata	OpenDataSoft
dcat:Dataset			
→ dct:title	title	name	title
→ dct:description	notes	description	description
→ dct:issued	metadata_created	createdAt	—
→ dct:modified	metadata_modified	viewLastModified	modified
→ dct:identifier	id	id	datasetid
→ dcat:keyword	tags	tags	keyword
→ dct:language	language	—	language
→ dct:publisher	organization	owner	publisher
→ dct:contactPoint	maintainer, author (-email)	tableAuthor	—
→ dct:accrualPeriodicity	frequency	—	—
→ dct:landingPage	url	—	—
→ dct:theme	—	category	theme
dcat:Distribution			
→ dct:title	resources.name	—	—
→ dct:issued	resources.created	—	—
→ dct:modified	resources.last_modified	—	—
→ dct:license	license_{id, title, url}	licenseId	license
→ dcat:accessURL	resources.url	*export URL*[a]	*export URL*[a]
→ dcat:downloadURL	resources.download_url	—	—
→ dct:format	resources.format	*export format*[a]	*export format*[a]
→ dct:mediaType	resources.mimetype	*export mime-type*[a]	*export mime-type*[a]
→ dct:byteSize	resources.size	—	—

[a]Socrata and OpenDataSoft offer data export in various formats via the API.

Figure 6: DCAT mapping of ODPW [26].

For ODPW, Neumaier et al. converted the metadata schemas of CKAN, Opendatasoft and Socrata using the **DCAT mapping** in figure 6. The ac-

[4]https://archiver.ai.wu.ac.at/api/v1/get/dataset/id/5e8634b5b511a400118c95af

companying Python code can be found in ODPW's GitHub repository[5] and could be used as a starting point for a DCAT mapping to be integrated into a future extension of our solution. However, a few changes to CKAN's schema will have to be considered: "resources.url" now contains the download URL instead of "resources.download_url" and "frequency" seemingly has been replaced by "update_frequency". In Opendatasoft's API v1, the mapping is still accurate, while in v2 and v2.1, only the field "datasetid" was renamed to "dataset_id". Opendatasoft's metadata schema differs the least from DCAT out of the three, as it is a subset of DCAT by default and the portal operator can activate the other DCAT fields[6]. Some updated fields can also be observed in the Socrata API, which is the only of the three that provides an easily accessible and complete metadata schema [37]. The field "viewLastModified" was replaced by "metadataUpdatedAt" and the fields "owner" and "tableAuthor" do not exist anymore, however this information can now be provided using "attribution" and within the nested set of "customFields". Also, there is no "licenseId", only a "license".

CKAN additionally offers a **DCAT extension** that enables the retrieval of metadata using DCAT[7]. We checked the availability of this extension and of the Turtle / RDF catalog for all CKAN portals on our new list. Our code is available in the experiments notebook and detailed results can be found in table 15 in the results chapter, but in short, only about a third of CKAN portals support these features. From these portals, DCAT metadata can be crawled and used directly without a conversion, however, the question is whether the benefits offset the increased effort and complexity of creating two separate processes for both types of CKAN portals and constantly keeping track of the extension support.

One contribution to portal monitoring that we can already provide thanks to our new system is an **analysis of the evolution** of ODPW's original portal list. We used the Data Portal Tracker to validate these portals' activity status and find out if they still have a functional CKAN, Opendatasoft or Socrata API. To get a better idea of how these portals have evolved over time, we compared our results to the original numbers from 2016 [26], the state of the list in 2019 [22] and a study from 2018 by Correa et al. [9] who performed a validation of the ODPW portals. Our findings and the detailed comparisons are presented in tables 13 and 14 and are discussed in the results chapter below.

[5]https://github.com/sebneu/portalwatch

[6]https://help.opendatasoft.com/faq-glossary/en/faq_index.html

[7]https://extensions.ckan.org/extension/dcat

5 Evaluation and Future Work

Now that the implementation details of the Data Portal Tracker have been thoroughly covered, we will evaluate the degree of goal attainment, present our initial results, report on the limitations that have become clear in the process and offer our suggestions for possible future work.

5.1 Evaluation of Goal Attainment

In the implementation chapter, we described all functional and technical aspects of our system that address the given goals and requirements in depth. To provide a much more compact summary, we will now present two tables that contain information on the attainment of goals and requirements. Starting with the functional goals we defined earlier, table 3 gives a concise overview of how each of them has been achieved.

Table 3: Attainment of the functional goals.

Code	**Description**
F.1	**Portal discovery and validation implemented**
F1.1	Semi-automated: search engine APIs, lists, manual additions
F1.2	Two-step validation: HTML markers, API functionality
F1.3	Two types of manual additions: domain, endpoint
F.2	**Open Data monitoring capabilities restored**
F.2.1	Dataset archiving extended to metadata
F.2.2	Mapping between datasets and metadata implemented
F.3	**ODPW reproducibility studies prepared**
F3.1	Evolution of ODPW's portal list analyzed
F3.2	Workflow for accessing metadata documented

Moving on, table 4 outlines the steps we have taken to design our system in ways that meet the given technical requirements.

Table 4: Attainment of the technical requirements.

Code	Description
T.1	**Resources managed**
T1.1	Hardware use minimized: search engine queries via APIs
T1.2	HTTP requests minimized: two-step validation, CSV assumption
T1.3	Information loss minimized: multi-step URL deduplication
T.2	**ODArchiver used and extended**
T.2.1	Crawler adds datasets/metadata to ODArchiver via its API
T.2.2	Mapping collection created in ODArchiver's MongoDB
T.2.3	Old datasets in ODArchiver corpus can be enriched
T.3	**Sustainable and well-documented architecture created**
T3.1	Code written in Python 3, requirements list provided
T3.2	Comments, docstrings, READMEs and notebooks used
T3.3	Thesis appendix contains detailed system documentation

5.2 Initial Results

After implementing and testing the Data Portal Tracker, we used it to create and validate a new portal list and subsequently count all datasets and resources of the portals. We will now present our results, collected from August to September 2023, and compare them to those of previous studies in the context of the respective system design. Moreover, we will show the evolution of the portals on ODPW's list from 2016 to 2023 as well as the outcomes of our DCAT extension availability check among CKAN portals.

First up, table 5 shows the results of our initial **portal discovery and validation** run which involved only a limited number of performed search queries. After collecting URLs from various sources, reducing them to their base URLs and removing duplicates, there are more than 5000 websites on the list. A little over 4000 of them were found to be active, the rest could not be reached or timed out. For now, only 4 portals with known API endpoints on subpages were manually added as an initial test of this feature. Our first validation step detected validation markers in the HTML code of 768 sites, while the second step found 717 portals with a working API. After the second

deduplication at the end of the pipeline, our list has **705 working portals**.

Table 5: Portal API validation results.

Total	**Active**	**Inactive**	**Subpage**	**Markers**	**Working**	**Unique**
5 539	4 380	1 159	4	768	717	705

Looking at the **platform-specifc results** in table 6, we can see that there are 355 unique Opendatasoft portals while 224 are based on CKAN and the other 126 use Socrata. We can also observe the difference between the number of sites with validation markers and the number of portals with working APIs which could be an indicator of the quality of our validation markers. Only the results for CKAN show a sizeable gap in this regard while for the other portal solutions, the marker-based validation had a precision of almost 100 percent as almost all sites suspected to have a working API actually had one.

Table 6: Validation results per API software.

Software	**Markers**	**Working**	**Unique**
CKAN	276	233	224
Opendatasoft	362	356	355
Socrata	130	128	126

Upon inspection of the specific HTML markers found on CKAN portals, we have realized that there are some obvious false positives, for example due to the "Powered by" marker which is an HTML element that can not only be followed by a CKAN logo on CKAN portals, but, among others, also by a logo of the Dataverse project[8] on portals using this software, e.g. the Harvard Dataverse or the Australian Data Archive.

However, these false positives are rather limited in number and a manual check of the relevant sites revealed that most cases can be attributed to non-standard API endpoints on subpages, unmaintained sites with broken APIs or instances where a portal has both validation markers and a working API, but each are located exclusively at different subdomains. An example of the latter is the Open Data portal of the City of York, which uses a subdomain for its working CKAN API[9], while validation markers can only be found in the HTML code of the front-end[10].

[8]https://dataverse.org
[9]https://data.yorkopendata.org
[10]https://www.yorkopendata.org

Subsequently, we moved on to counting and aggregating the number of datasets and resources on each portal. As shown in table 7, we found more than **3 million datasets** and almost **7 million resources** or datasets in export formats supported by our crawling functions, respectively. To know how much data the ODArchiver will have to index when our system crawls all of these portals, we can estimate the number of unique metadata pages to be 3 million, as resources do not have separate metadata. Therefore, based on the number of resources and supported datasets plus the related metadata, we can estimate the total sum to be around 10 million dataset and metadata entities. However, we have to keep in mind that the ODArchiver enforces a file size limit and does not index any datasets/resources exceeding it. Still, for comparison, the ODArchiver corpus currently consists of less than 1.2 million datasets/resources, which indicates that our system will extend the institute's portal monitoring capabilities significantly. Note that the much higher number of *files* presented on the ODArchiver webpage includes all versions of a dataset or resource[11]. Also, when checking a large list of portals, not every single one will always be reachable, thus there is a slight difference between the number of portals in table 6 and the number of portals on which the data in table 7 is based. In total, 692 portals responded to our requests during this check (349 Opendatasoft, 218 CKAN, 125 Socrata).

Table 7: Datasets, supported datasets and resources per API software.

Software	**Datasets**		**Resources/Supported**	
	Total	**Average**	**Total**	**Average**
CKAN	2 729 379	12 520	6 680 682	30 645
Opendatasoft	90 636	260	90 636	260
Socrata	186 603	1 493	64 097	513
Total	**3 006 618**	**4 345**	**6 835 415**	**9 878**

For each portal software, we extracted the detailed results per portal and compiled the largest 5 portals in terms of resources or supported datasets, as this is the most relevant metric for our system. The resulting numbers are sorted in descending order and are presented in tables 8, 9 and 10. Starting with the **largest CKAN portals**, the top 5 are accounting for 78 percent of all resources on more than 200 CKAN portals, with the top 2 alone making up 69 percent, highlighting a strong imbalance in the size of Open Data portals.

[11] https://archiver.ai.wu.ac.at/stats

Table 8: Largest CKAN portals by resources.

Portal	Datasets	Resources
https://data.amerigeoss.org	647 468	3 064 339
https://catalog.data.gov	250 615	1 554 272
https://data.gov.ua	29 243	225 757
https://ckan.publishing.service.gov.uk	56 145	213 544
https://scmb-ckan-dev.research.dc.uq.edu.au	1 343	182 581

Looking at the **largest Opendatasoft portals**, we can immediately find a potential duplicate as the numbers in spots 3 and 4 are matching exactly and indeed, both domains can be attributed to the French administrative region of Occitania. Our understanding of the Opendatasoft Data Hub is that every single dataset published on any Opendatasoft portal is automatically also made available on the Data Hub[12]. Naturally, it is therefore ranked number 1, however it only lists 35 percent of all Opendatasoft datasets we discovered as opposed to 50 percent. Thus, either the Data Hub does not fully cover every portal or there is a much higher level of duplication between portals than we have confirmed so far.

Table 9: Largest Opendatasoft portals by supported datasets.

Portal	Datasets	Supported
https://data.opendatasoft.com	31507	31507
https://smartregionidf.opendatasoft.com	8777	8777
https://data.laregion.fr	1601	1601
https://occitanie.opendatasoft.com	1601	1601
https://analyzejerseycity.opendatasoft.com	1256	1256

In the table of the **largest Socrata portals**, there is also an obvious duplicate as portals number 1 and 2 have the same dataset count. As it turns out, both domains belong to the Open Data portal of Colombia. For future extensions to the Data Portal Tracker, such cases should be addressed, whether by manually keeping a list of domains or by finding an automated solution that is carefully calibrated so as not to lead to information loss.

We will now **compare the results of multiple studies** on data portal discovery that we mentioned in the literature chapter. Table 11 lists how

[12]https://www.opendatasoft.com/en/blog/what-is-the-opendatasoft-data-hub/

Table 10: Largest Socrata portals by supported datasets.

Portal	Datasets	Supported
https://www.datos.gov.co	31 484	6 786
https://colombia-mintic.data.socrata.com	31 484	6 786
https://data.ny.gov	5 681	4 172
https://dati.lombardia.it	5 542	3 376
https://bronx.lehman.cuny.edu	4 319	3 209

many CKAN, Opendatasoft and Socrata portals were found in the ODPW study by Neumaier et al., in three consecutive studies by Correa et al. and in the first run of our Data Portal Tracker. In total, our number of identified APIs is similar to those of the studies in which Common Crawl archives were used. We found more Opendatasoft portals and fewer CKAN and Socrata portals, which might have been impacted by our sources, particularly the web lists, and would be an interesting issue to investigate in future work.

Table 11: Portal API validation results of different studies.

Study	Year	CKAN	Opendatasoft	Socrata	Total
Neumaier et al. [26]	2016	148	11	102	261
Correa et al. [9]	2018	185	39	132	356
Correa & Silva [8]	2019	351	167	201	719
Correa et al. [7]	2020	439	143	255	837
Data Portal Tracker	2023	224	355	126	705

Details on each study's methodology were already covered in the literature chapter, but to put the results in context, table 12 offers essential information regarding the chosen data sources at a glance.

Table 12: Portal API sources of different studies.

Study	Manual	Lists	Common Crawl	Search API
Neumaier et al. [26]	✓			
Correa et al. [9]		✓		
Correa & Silva [8]			✓	
Correa et al. [7]			✓	
Data Portal Tracker	✓	✓		✓

Table 13 shows the **evolution of portals** with working APIs on ODPW's list from 2016 to 2023. The initial data from 2016 was taken from the original ODPW study, while the data point for 2018 is based on results we found in one of the mentioned studies by Correa et al. ODPW's GitHub repository contains a portals.ttl file updated in 2019 which provided us with the data for that year. We created the results for 2023 by validating the latest version of the ODPW list using the Data Portal Tracker.

Table 13: Evolution of working portal APIs on the ODPW list.

Study	Year	CKAN	Opendatasoft	Socrata	Total
Neumaier et al. [26]	2016	148	11	102	261
Correa et al. [9]	2018	85	10	77	172
Neumaier et al. [22]	2019	118	11	65	194
Data Portal Tracker	2023	61	9	46	116

In table 14, "All" refers to the raw portal list and "Relevant" denotes the list after keeping only portals marked as CKAN, CKANDCAT, Opendatasoft or Socrata, changing CKANDCAT to CKAN by truncating the "/catalog.ttl" path and removing the resulting duplicates. "(In-)active" means that the site could (not) be reached and "Working" counts all portals with working APIs.

Table 14: Detailed validation results for the ODPW list.

Study	All	Relevant	Inactive	Active	Working
Neumaier et al. [26]	261	261	0	261	261
Correa et al. [9]	-	267	37	230	172
Neumaier et al. [22]	278	267	76	191	191
Data Portal Tracker	278	267	90	177	116

Socrata portals show a gradual decline over the years, while the few Opendatasoft portals on the list remained almost constant. Only the CKAN portals show a sudden increase in 2019, also leading to an overall increase of working portals compared to 2018. However, one limitation of this comparison is the uncertainty regarding methodological differences. Particularly, the data from 2019 does not differentiate between active and working portals. It is not fully clear whether the ODPW maintainers exchanged certain inactive portals for active ones over time, whether any broken URLs were corrected or whether the "active" label from the list can even be equated to "working",

as we do not know the exact procedures regarding the list updates. Additionally, Correa et al. might have had a similar approach regarding filtering the raw list - in this case, the value for "All" for 2018 would be the same as that for 2023 if the list did not change from 2018 to 2019. The data we are most confident about is found in the columns on inactive and active portals in table 14, showing a **steady decrease in active portals**, with 68 percent of portals still being active 7 years after the first study. Fully operational portals that did not change the path of their API endpoint or implement an API software other than CKAN, Opendatasoft or Socrata are even less common, as we could only find a working API on 44 percent of portals.

Table 15: CKAN portals with enabled DCAT extension.

Total	**Checked**	**DCAT extension**	**TTL catalog**
224	217	77	68

Finally, we investigated which degree of availability of CKAN's DCAT extension can be assumed in future work by counting the CKAN portals on our list whose API response showed the extension to be enabled. Additionally, we also checked whether a Turtle / RDF catalog was present. Table 15 shows that only 35 percent of portals that successfully responded to our request made use of the DCAT exension, while 31 percent featured a Turtle / RDF catalog.

5.3 Limitations and Future Work

During development, testing and evaluation of the Data Portal Tracker, we encountered a number of weaknesses of the system, came up with possible extensions and improvements and thought of several ways in which the newly available tools and data could be used for further analyses and research.

First up, the most significant limitation is a **bug in the ODArchiver** that will hopefully be fixed soon. All functions of the ODArchiver required for development and testing of our system work fine up until and including the point where it indexes the dataset and metadata, which successfully are assigned an ID. Additionally, regular crawls of existing datasets in ODArchiver's corpus seem to work, as new versions of datasets are still added to the MongoDB. For newly added datasets and metadata, however, the timestamps indicate that crawls are performed, but the downloading of data does not work as the actual files never end up in the MongoDB.

After this bug is fixed and the datasets and metadata of all new portals have been crawled, the logical next step will be to implement the mentioned DCAT mapping, fully restore ODPW's capabilities and resume calculating **metadata scores**, both for ODPW's portals in a reproducibility study and for all new portals we discovered.

Regarding the crawling, it could also be worth investigating the possible issue of the ODArchiver's **crawling being out of sync** between datasets and the related metadata in the future due to the ODArchiver's dynamic crawling frequency. First, it should be analyzed how much they are out of sync and if necessary, actions should be taking to synchronize the crawls of datasets and the associated metadata, for example using the mapping information. Of course, this would require a modification of the ODArchiver's code.

We have documented the required steps for **adding another portal software** to the system in detail in the appendix, and one such Open Data platform software that could be of interest for a future addition is ArcGIS[13]. Piveau, which powers data.europa.eu, could also be considered. However, when extending the Data Portal Tracker by adding support for data.europa.eu or other sites that aggregate datasets from many individual portals, a strategy should first be defined to avoid duplicates of datasets that originate from sources that we already support and index.

Also, a method for **detecting duplicates** in cases where the same datasets are offered for download on multiple portals should be defined and implemented. In such instances, the same datasets might be hosted on multiple different portals or on the same portal using multiple domain names (such as the covered examples of Colombia and the French region of Occitania). Mitigation might involve analyzing the file contents and manually examining any matching numbers of datasets on multiple portals.

Concerning the portal discovery pipeline, highly interesting insights might be generated by researching the detailed **impact of each data source on the final list** of portals by running the complete list creation and validation process once per source while temporarily removing all other data inputs within the `create_list` function and comparing the results.

Currently, Crawley is operated by manually calling the script on the command line and passing the search parameters. In the future, this could be fully automated by defining a fixed set of search terms and writing a function that lets the system request the queries automatically upon execution.

[13]https://www.esri.com/en-us/arcgis/products/arcgis-open-data

Checking the activity status of a portal could be improved. With the current approach, some active portals are not included, but rather labeled as inactive. For example, the data portal of New York City has a working Socrata API endpoint[14], but accessing the base URL[15] returns the HTTP code 403 (Forbidden) when requesting it via the Python script. This is because the front-end of the portal uses a different domain[16] which users accessing the API's base URL via their browser are redirected to. More broadly, the general approach to activity checking the websites in the pipeline could be simplified and improved. For example, the `add_prefixes` function which requests the base URL to check website activity could be called in the subsequent list validation function. Instead of calling these two functions in separate steps, integrating them would allow the activity check to also take advantage of the validation function's already existing ability to retry failed sites. Not only would very short outages no longer lead to websites being marked inactive and skipped in the validation, but this change could also contribute to addressing the varying availability of certain portals encountered in testing and would simplify the regular reevaluation of portals that were unavailable at an earlier point.

Going forward, improvements in this area could contribute to a more detailed analysis on why specific portals are still active and can be reached, but do not have a working API at the known and tested path anymore. For this purpose, it would be helpful to capture and analyze the status code and the complete exception instead of just the type.

Fine-tuning the validation markers could help reduce the number of marker-related false positives. For example, there are certain domains that appear repeatedly in search results and might contain validation markers, but almost certainly do not host a valid Open Data portal using CKAN, Opendatasoft or Socrata. Subdomains like *github.com, *github.io, *softonic.com and websites like https://ckan.org could be removed from the list or even passed as parameters to the search engine queries to be excluded from the beginning.

While functionality to manually add portals featuring an **API endpoint on a non-standard subpage** has been implemented in the Data Portal Tracker, carrying out the associated tasks is now up to the future users of the system. To search for more portals like these, the CSV file containing the validation results can be used as a starting point. Relevant cases are portals on which API markers were found, but for which no working API could be

[14]https://data.cityofnewyork.us/api/views/metadata/v1
[15]https://data.cityofnewyork.us
[16]https://opendata.cityofnewyork.us

verified. After identifying a portal, its HTML code should be accessed to look for portal software indicators and the JSON file containing the markers found on the site during the validation should be consulted to determine if the validation markers need fine-tuning. In the experiments notebook, we have provided code in the section "Checking false positives of marker validation" to help identify relevant cases.

6 Conclusion

Data published by governments or companies free of charge and without usage restrictions in Open Data portals can have significant economic value and positive societal impacts. However, to achieve these benefits, certain characteristics like usability and discoverability play a vital role and are facilitated by high-quality metadata. Open Data monitoring solutions are keeping track of these aspects and are compiling lists of available portals. However, portal lists are typically either created manually or use methods that are extremely resource-intensive and not dynamic, while many metadata monitoring platforms, including WU's ODPW, are no longer fully operational.

In this thesis, we presented the Data Portal Tracker, an integrated solution for portal discovery, validation and monitoring as well as dataset and metadata crawling. It features a semi-automated system to dynamically discover Open Data portals by querying search engines via the tool Crawley and combines the results with web lists and manual additions. After deduplicating the websites and checking their activity status and HTTPS support, all potential portals are validated in a two-step process based on HTML markers and API functionality testing. Our system was built with support for the portal software options CKAN, Opendatasoft and Socrata, but can be extended by following the steps documented in the appendix of the thesis. Additionally, the Data Portal Tracker crawls all dataset and metadata URLs from each portal and sends them to the ODArchiver, a data archiving and versioning tool whose prior functionality was extended: instead of only indexing datasets, metadata is now also included.

Our initial results show 705 functional portals using CKAN, Opendatasoft or Socrata, a significant increase over ODPW's 261 identified portals and comparable to more resource-intensive and less dynamic solutions found in literature. We further analyzed the evolution of the ODPW portals since 2016 and found that 68 percent of sites are still active and 44 percent still feature a working API based on one of the three studied software solutions. Archiving all datasets, resources and metadata of all portals on our new list will be the basis for a future metadata rating system to replace the related features of ODPW and will lead to the ODArchiver crawling 10 million URLs, vastly expanding its corpus of currently 1.2 million records.

While the Data Portal Tracker is a dynamic and extensible solution, makes use of the highly scalable ODArchiver and provides interesting first results, there is still room for improvement in future work. By providing detailed technical documentation, we want to facilitate extensions to our system that add support for additional portal software or implement metadata quality rating mechanisms.

References

[1] Directive (EU) 2019/1024 of the European Parliament and of the Council of 20 June 2019 on open data and the re-use of public sector information (recast). *Official Journal of the European Union, L 172*, 62:56–83, June 2019.

[2] Commission Implementing Regulation (EU) 2023/138 of 21 December 2022 laying down a list of specific high-value datasets and the arrangements for their publication and re-use (Text with EEA relevance). *Official Journal of the European Union, L 019*, 66:43–75, January 2023.

[3] Husam Barham, Marina Dabic, Tugrul Daim, and Dara Shifrer. The role of management support for the implementation of open innovation practices in firms. *Technology in Society*, 63:101282, November 2020.

[4] Omar Benjelloun, Shiyu Chen, and Natasha Noy. Google Dataset Search by the Numbers. In Jeff Z. Pan, Valentina Tamma, Claudia d'Amato, Krzysztof Janowicz, Bo Fu, Axel Polleres, Oshani Seneviratne, and Lalana Kagal, editors, *The Semantic Web – ISWC 2020*, volume 12507, pages 667–682. Springer International Publishing, Cham, 2020. Series Title: Lecture Notes in Computer Science.

[5] BuiltWith. Websites using CKAN. URL: https://trends.builtwith.com/websitelist/CKAN. Last accessed: 14 March 2023.

[6] Soumen Chakrabarti, Martin Van Den Berg, and Byron Dom. Focused crawling: a new approach to topic-specific Web resource discovery. *Computer Networks*, 31(11-16):1623–1640, May 1999.

[7] Andreiwid Sheffer Correa, Alencar Melo Jr., and Flavio Soares Correa Da Silva. A deep search method to survey data portals in the whole web: toward a machine learning classification model. *Government Information Quarterly*, 37(4):101510, October 2020.

[8] Andreiwid Sheffer Correa and Flavio Soares Correa Da Silva. Laying the foundations for benchmarking open data automatically: a method for surveying data portals from the whole web. In *Proceedings of the 20th Annual International Conference on Digital Government Research*, pages 287–296, Dubai United Arab Emirates, June 2019. ACM.

[9] Andreiwid Sheffer Correa, Pär-Ola Zander, and Flavio Soares Correa Da Silva. Investigating open data portals automatically: a methodology

and some illustrations. In *Proceedings of the 19th Annual International Conference on Digital Government Research: Governance in the Data Age*, pages 1–10, Delft The Netherlands, May 2018. ACM.

[10] Horia-Stefan Dinu and Nicolas Ferranti. GitLab - Portalwatch API, 2021. URL: https://git.ai.wu.ac.at/ferranti/portalwatch_api. Last accessed: 8 September 2023.

[11] Daniil Dobriy and Axel Polleres. Crawley: A Tool for Web Platform Discovery. In *Proceedings of the 22nd International Semantic Web Conference*, 2023.

[12] Directorate-General for the Information Society and Media European Commission. *Commercial exploitation of Europe's public sector information: executive summary*. Publications Office of the European Union, Luxembourg, 2000.

[13] Open Knowledge Foundation. API guide - CKAN 2.10.1 documentation. URL: https://docs.ckan.org/en/2.10/api/. Last accessed: 10 September 2023.

[14] Open Knowledge Foundation. Data Portals. URL: https://dataportals.org. Last accessed: 25 May 2023.

[15] Christian Philipp Geiger and Jörn Von Lucke. Open Government and (Linked) (Open) (Government) (Data). *JeDEM - eJournal of eDemocracy and Open Government*, 4(2):265–278, December 2012.

[16] Marijn Janssen, Yannis Charalabidis, and Anneke Zuiderwijk. Benefits, Adoption Barriers and Myths of Open Data and Open Government. *Information Systems Management*, 29(4):258–268, September 2012.

[17] Fabian Kirstein, Anton Altenbernd, Sonja Schimmler, and Manfred Hauswirth. A Decentralised Persistent Identification Layer for DCAT Datasets. In *Companion Proceedings of the ACM Web Conference 2023*, pages 1424–1427, Austin TX USA, April 2023. ACM.

[18] Fabian Kirstein, Kyriakos Stefanidis, Benjamin Dittwald, Simon Dutkowski, Sebastian Urbanek, and Manfred Hauswirth. Piveau: A Large-Scale Open Data Management Platform Based on Semantic Web Technologies. In Andreas Harth, Sabrina Kirrane, Axel-Cyrille Ngonga Ngomo, Heiko Paulheim, Anisa Rula, Anna Lisa Gentile, Peter Haase, and Michael Cochez, editors, *The Semantic Web*, volume 12123,

pages 648–664. Springer International Publishing, Cham, 2020. Series Title: Lecture Notes in Computer Science.

[19] Erik Lakomaa and Jan Kallberg. Open Data as a Foundation for Innovation: The Enabling Effect of Free Public Sector Information for Entrepreneurs. *IEEE Access*, 1:558–563, 2013.

[20] Martin Lnenicka and Anastasija Nikiforova. Transparency-by-design: What is the role of open data portals? *Telematics and Informatics*, 61:101605, August 2021.

[21] Johann Mitlohner, Sebastian Neumaier, Jurgen Umbrich, and Axel Polleres. Characteristics of Open Data CSV Files. In *2016 2nd International Conference on Open and Big Data (OBD)*, pages 72–79, Vienna, August 2016. IEEE.

[22] Sebastian Neumaier. GitHub - Open Data Portal Watch, 2019. URL: https://github.com/sebneu/portalwatch. Last accessed: 26 August 2023.

[23] Sebastian Neumaier. *Semantic enrichment of open data on the Web - or: how to build an open data knowledge graph.* PhD thesis, TU Wien, 2019.

[24] Sebastian Neumaier, Lörinc Thurnay, Thomas J. Lampoltshammer, and Tomá Knap. Search, Filter, Fork, and Link Open Data: The ADEQUATe platform: data- and community-driven quality improvements. In *Companion of the The Web Conference 2018 on The Web Conference 2018 - WWW '18*, pages 1523–1526, Lyon, France, 2018. ACM Press.

[25] Sebastian Neumaier and Jürgen Umbrich. Measures for Assessing the Data Freshness in Open Data Portals. In *2016 2nd International Conference on Open and Big Data (OBD)*, pages 17–24, Vienna, August 2016. IEEE.

[26] Sebastian Neumaier, Jürgen Umbrich, and Axel Polleres. Automated Quality Assessment of Metadata across Open Data Portals. *Journal of Data and Information Quality*, 8(1):1–29, November 2016.

[27] Sebastian Neumaier, Jürgen Umbrich, and Axel Polleres. Lifting Data Portals to the Web of Data. In *Workshop Proceedings of the 10th Workshop on Linked Data on the Web (LDOW2017)*, Perth, Australia, 2017. CEUR Workshop Proceedings.

[28] University of Oxford. FAIRsharing | Databases. URL: https://fairsharing.org/search?fairsharingRegistry=Database. Last accessed: 14 March 2023.

[29] Karlsruhe Institute of Technology. Registry of Research Data Repositories. URL: https://www.re3data.org. Last accessed: 14 March 2023.

[30] Publications Office of the European Union. Metadata Quality | Official Portal for EU Data. URL: https://data.europa.eu/mqa/?locale=en. Last accessed: 26 August 2023.

[31] OpenDataMonitor. European Data Catalogues Overview. URL: https://opendatamonitor.eu/frontend/web/index.php?r=datacatalogue/list. Last accessed: 19 September 2023.

[32] OpenDataMonitor. FAQ | Open Data Monitor Knowledge Base. URL: http://knowhow.opendatamonitor.eu/help/. Last accessed: 28 August 2023.

[33] Opendatasoft. Open Data Inception. URL: https://data.opendatasoft.com/explore/dataset/open-data-sources@public. Last accessed: 27 August 2023.

[34] Opendatasoft. Opendatasoft's Explore API Reference Documentation (v2.1). URL: https://help.opendatasoft.com/apis/ods-explore-v2/. Last accessed: 10 September 2023.

[35] Opendatasoft. Search API v1 documentation. URL: https://help.opendatasoft.com/apis/ods-search-v1/. Last accessed: 10 September 2023.

[36] Per Runeson, Thomas Olsson, and Johan Linåker. Open Data Ecosystems — An empirical investigation into an emerging industry collaboration concept. *Journal of Systems and Software*, 182:111088, December 2021.

[37] Socrata. Metadata API. URL: https://socratametadataapi.docs.apiary.io. Last accessed: 10 September 2023.

[38] Tyler Technologies. API Endpoints | Socrata. URL: https://dev.socrata.com/docs/endpoints.html. Last accessed: 10 September 2023.

[39] World Wide Web Consortium (W3C). Data Catalog Vocabulary (DCAT) - Version 2, February 2020. URL: https://www.w3.org/TR/vocab-dcat-2. Last accessed: 5 September 2023.

[40] Thomas Weber, Johann Mitöhner, Sebastian Neumaier, and Axel Polleres. ODArchive – Creating an Archive for Structured Data from Open Data Portals. In Jeff Z. Pan, Valentina Tamma, Claudia d'Amato, Krzysztof Janowicz, Bo Fu, Axel Polleres, Oshani Seneviratne, and Lalana Kagal, editors, *The Semantic Web – ISWC 2020*, volume 12507, pages 311–327. Springer International Publishing, Cham, 2020. Series Title: Lecture Notes in Computer Science.

[41] Bianca Wentzel, Fabian Kirstein, Torben Jastrow, Raphael Sturm, Michael Peters, and Sonja Schimmler. An Extensive Methodology and Framework for Quality Assessment of DCAT-AP Datasets. In Ida Lindgren, Csaba Csáki, Evangelos Kalampokis, Marijn Janssen, Gabriela Viale Pereira, Shefali Virkar, Efthimios Tambouris, and Anneke Zuiderwijk, editors, *Electronic Government*, volume 14130, pages 262–278. Springer Nature Switzerland, Cham, 2023. Series Title: Lecture Notes in Computer Science.

[42] Anneke Zuiderwijk, Marijn Janssen, Kostas Poulis, and Geerten Van De Kaa. Open data for competitive advantage: insights from open data use by companies. In *Proceedings of the 16th Annual International Conference on Digital Government Research*, pages 79–88, Phoenix Arizona, May 2015. ACM.

A Documentation

As a successor to *Open Data Portal Watch (ODPW)* [26], the Data Portal Tracker aims to semi-automatically create, validate and regularly update a comprehensive list of Open Data portals, crawl the URLs of all datasets and the associated metadata on these portals and add them to the *Open Dataset Archiver (ODArchiver)* [40] for downloading, periodic crawling and version tracking. It consists of the core *Data Portal Tracker* functionality and an adapted version of Daniil Dobriy's search engine crawling tool *Crawley* [11] and connects to the *ODArchiver's* API and MongoDB.

The subsequent documentation chapters and the extensive comments in the code aim to provide the reader with a thorough understanding of the tool's functionality and will enable them to try out, deploy, improve and extend the code. A concise version of the information presented in this section can be found in the README files in the Git repository[17] - see README.md and crawley-lite/README.md.

Even though testing has shown no major issues, we recommend performing manual sanity checks whenever results seem odd. If you want to improve this tool and need a starting point, please see the limitations and future work chapter of the thesis. Requirements for extracting metadata from the ODArchiver as a first step towards a metadata rating system can be found in section 4.4 in the main part of the thesis. Additionally, chapter A.8 below outlines the technical steps to add support for other portal software options.

Chapter A.1 shows the necessary steps to deploy the system.

Chapter A.2 covers search engine portal discovery using Crawley.

Chapter A.3 describes the list creation and validation pipeline.

Chapter A.4 presents the four crawling scripts.

Chapter A.5 explains how the crawling scripts connect to the ODArchiver.

Chapter A.6 outlines the helper functions.

Chapter A.7 gives an overview of the experiments that we carried out.

Chapter A.8 provides details on how to extend the system.

[17] https://git.ai.wu.ac.at/h1613073/data_portal_tracker

A.1 Deploying the system

Before deploying the Data Portal Tracker, first install the Python libraries listed in the requirements.txt file in the project's root directory.

To use the full functionality of the system and be able to connect to the ODArchiver's production MongoDB and use the method of the ODArchiver API that posts resources, request the .env file from the institute's system administrators and save it in the project's root directory. While the repository includes an .env_example file, it does not contain the API secret and the MongoDB connection strings.

The ODArchiver's MongoDB uses a Kubernetes cluster and is currently running on three nodes, that's why there are three production connection strings in the .env file and multiple try/except blocks in the __init__ method of the ArchiverConnector class that try out these strings. A virtual machine provided by the institute is needed to connect to these nodes, each of which has a port that was opened for this purpose.

Whenever the ArchiverConnector class, which is defined in the data_portal_tracker/archiver_connector.py script, is instantiated, for example after importing it into the data_portal_tracker/portal_crawler.(ipynb|py) script, the argument "mode" must be set to "production" instead of "local".

Checklist:

- Install Python libraries
- Save .env file to project root
- Use "[...]ai.wu.ac.at" virtual machine
- Call ArchiverConnector(mode = "production")

For brevity, some of the subsequent Python function call examples will only be shown in the short form used within a script (see listing 1), but the functions can of course also be executed on the command line (see listing 2).

```
1 crawl_ckan(portal_list, "data/portal_statistics_ckan.csv")
```

Listing 1: Function call within a script

```
1 python3 -c 'from portal_crawler import *;
  ↪  crawl_ckan(portal_list, "data/portal_statistics_ckan.csv")'
```

Listing 2: Function call on the command line

A.2 Search engine portal discovery

Path: crawley-lite/.*

The tool Crawley, which we use for **portal discovery** via search engine APIs, was provided by Daniil Dobriy [11] and subsequently adapted and simplified for our use case. Also, parts of our documentation related to the tool were taken over and edited.

When using Crawley, make sure that there are SerpAPI keys in the crawley-lite/keys.txt file. If there are none, register for a free account and add keys to the file. In order to perform a Google search for "Open Data Portal", use the command line to navigate to the crawley-lite directory and execute the crawley-lite/crawley-lite.py script:

```
python3 crawley-lite.py --query "Open Data Portal" --engine
↪ Google --count 100  --offset 0
```

Listing 3: Portal discovery: basic search engine query

For Google searches, there are up to 100 results per query - the number of results to be returned is controlled by the parameter --count, while the number of results to be skipped is controlled by the parameter --offset. In pagination terms, when showing 100 results per page, you can skip to page 2 by specifying an offset of 100, to page 3 with an offset of 200 and so on.

```
python3 crawley-lite.py --query "Open Data Portal" --engine
↪ Google --count 100  --offset 100
```

Listing 4: Portal discovery: search engine query with offset

Just like on the web interface, quotes can be used for exact matches (and must be escaped when using the same type of quotes for the whole search query) and search operators can restrict the results to certain domains or exclude domains.

```
python3 crawley-lite.py --query "site:*.opendatasoft.com \"Open
↪ Government Data\" -site:data.opendatasoft.com" --engine
↪ Google --count 100  --offset 0
```

Listing 5: Portal discovery: search engine query with search operators

Search results returned by the API are saved to JSON files in the crawley-lite/results folder. These files are later used as input files in the portal_handler script in the data_portal_tracker project directory, which extracts all of the organic result URLs and adds them to the list creation and portal validation pipeline.

Listing 6 shows the first organic search result in one of the JSON files mentioned above and gives an idea of the additional, currently unused data that is available for potential further refinement of the system in the future:


```
{
    "position": 1,
    "title": "Data.gov CKAN API - Catalog",
    "link":
    ↪ "https://catalog.data.gov/dataset/data-gov-ckan-api",
    "displayed_link": "https://catalog.data.gov > dataset >
    ↪ data-gov-ckan-api",
    "date": "Nov 10, 2020",
    "snippet": "The data.gov catalog is powered by CKAN, a
    ↪ powerful open source data platform that includes a robust
    ↪ API. Please be aware that data.gov and ...",
    "snippet_highlighted_words": [
        "powered by CKAN"
    ],
    "rich_snippet": {
        "bottom": {
            "detected_extensions": {
                "data_update_frequency_r_pm": 1,
                "metadata_created_date_november": 10
            },
            "extensions": [
                "Data Update Frequency: R/P1M",
                "Metadata Created Date: November 10, 2020"
            ]
        }
    },
    "about_this_result": {
        "keywords": [
            "powered",
            "by",
            "ckan"
        ],
```

```
        "languages": [
            "English"
        ],
        "regions": [
            "United States"
        ]
    },
    "cached_page_link":
    "https://webcache.googleusercontent.com/search?q=cache:
    L-lWAlt2ab8J:https://catalog.data.gov/dataset/
    data-gov-ckan-api&cd=10&hl=en&ct=clnk&gl=us",
    "related_pages_link":
    "https://www.google.com/search?q=related:
    https://catalog.data.gov/dataset/data-gov-ckan-api",
    "source": "data.gov"
}
```

Listing 6: Portal discovery: search result example

More details about our adapted version of Crawley can be found in the crawley-lite/README.md file. The original code is also available on GitHub[18].

[18] https://github.com/semantisch/crawley

A.3 Portal list creation and validation

Path: data_portal_tracker/portal_handler.(ipynb|py)

Our **portal handler** uses a multi-step pipeline to create a list of portals from various sources and validate the websites on the list. It allows manual additions at two different points.

Currently, files created by functions are saved to data_portal_tracker/data, but for all files whose paths are passed to functions as arguments, this can be changed. When rerunning the script's functions to create an updated list, we recommend creating a new (sub-)folder for every update and change the paths passed to the functions as arguments accordingly. In this way, no existing data is overwritten and the evolution of data portals can be analyzed.

Log files containing failure or success information that are generated during the execution of functions are stored in the data_portal_tracker/logs directory. To change this, the function definitions would have to be edited. Log files that are automatically created by functions are stored in CSV format and include the function name and a timestamp in their name - for example: crawl_socrata_2023-08-15_16_52_42_fail.csv. In addition, we manually saved printed output of functions to files that include the function name in their name and a have .log file extension - for example: validate_list.log.

A.3.1 Extract search results

Once the search engine portal discovery component has delivered some results, we can use the function extract_search_results to extract the URLs of organic search results from the saved JSON files. This function includes code for looping through the search results adapted from Crawley.

When calling the function, two arguments are required:

- search_results_folder (string): the path of the folder in the crawley-lite directory containing the search results
- output_file (string): the path of the CSV file to be exported

Here is how we called the function for the first run:

```
extract_search_results(
    search_results_folder = "../crawley-lite/results",
    output_file = "data/0_search_results.csv")
```

Listing 7: Portal handler: extract search results

A.3.2 Create a portal list

Next, the function create_list is used to create an initial list of portal URLs based on multiple sources. This function contains the first of two options for manually adding portals. All URLs added here will go through the entire deduplication and validation process without bypassing any steps. Currently, additions are made by extending the array additional_portals in the function definition, however, it might be useful to do this via a function parameter in the future (just like we already implemented in the add_api_endpoints function, the second option for manual additions).

When calling the function, two arguments are required:

- search_results_file (string): the path of the CSV input file containing search result URLs in a column "url"
- output_file (string): the path of the CSV file to be exported

Here is how we called the function for the first run:

```
create_list(
    search_results_file = "data/0_search_results.csv",
    output_file = "data/1_initial_portals.csv")
```

Listing 8: Portal handler: create list

A.3.3 Deduplicate the portal list

The function remove_duplicates deduplicates a list of URLs by removing unwanted characters, reducing each URL to its base URL, truncating the HTTP(S) protocol prefix and finally dropping all duplicates.

When calling the function, two arguments are required:

- initial_portals_file (string): the path of the CSV input file containing initial portal URLs in a column "url"
- output_file (string): the path of the CSV file to be exported

Here is how we called the function for the first run:

```
remove_duplicates(
    initial_portals_file = "data/1_initial_portals.csv",
    output_file =  "data/2_deduplicated_portals.csv")
```

Listing 9: Portal handler: remove duplicates

A.3.4 Add manually validated API endpoints

The function add_api_endpoints addresses two issues:

- Some portals have API endpoints that use a non-standard path (e.g. /catalog/api instead of /api).
- The second validation step of the validate_list function (see section A.3.6) only checks the API functionality of portals for which HTML markers were found in the first validation step.

Therefore, the function add_api_endpoints can be used to add portals which use a custom path or are known to support a certain API, but do not have any HTML markers. If they are added in this way, the first validation step of the validate_list function is bypassed and the portals will end up on the final portal list. Also note that portals added like this will be counted as "suspected" for their respective portal software in the validation statistics - thus, the number of "suspected" portals is not fully equivalent to the number of sites for which portal HTML markers were found.

When calling the function, three arguments are required:

- manual_api_additions_file (string): the path of the CSV input file containing API base URLs without "/api/..." in a column "url", e.g. "data.gv.at/katalog", and the API software name or "Unknown" in a column "manually_checked_api"
- deduplicated_portals_file (string): the path of the CSV input file containing deduplicated portal URLs in a column "url"
- output_file (string): the path of the CSV file to be exported

Here is how we called the function for the first run:

```
1 add_api_endpoints(
2     manual_api_additions_file =
      ↪ "data/manual_api_additions.csv",
3     deduplicated_portals_file =
      ↪ "data/2_deduplicated_portals.csv",
4     output_file = "data/3_extended_portals.csv")
```

Listing 10: Portal handler: add API endpoints

A.3.5 Add protocol prefixes and activity status

For each entry in the input list, the function requests the URL with HTTPS, then falls back to HTTP in case of an exception or a response code indicating failure. Information about the supported protocol and the website activity status is added to each row in the DataFrame and the enriched list is exported to a CSV file.

When calling the function, two arguments are required:

- extended_portals_file (string): the path of the CSV input file containing portal URLs in a column "url"
- output_file (string): the path of the CSV file to be exported

Here is how we called the function for the first run:

```
add_prefixes(
    extended_portals_file = "data/3_extended_portals.csv",
    output_file = "data/4_prefixed_portals.csv")
```

Listing 11: Portal handler: add prefixes

A.3.6 Validate the list

To identify portals that use one of our supported API software solutions (CKAN, Opendatasoft, Socrata), the function validate_list iterates over the input portal list, validates that the portals use a relevant catalog software and exports the validation results. Portal software validation is a two-step process: First, the function searches a portal's HTML code for specific validation markers that indicate the use of a certain software. If HTML markers are found or if a portal was previously added to the list via the function add_api_endpoints, the function checks whether the suspected API is working. This function includes code for HTML validation and related JSON exporting adapted from Crawley. The rules for the marker-based validation are located in the crawley-lite/config.json file.

Since there is always at least a small number of websites that are unavailable at a given time, we implemented an optional mode that allows you to retry the failed portals without starting from scratch. To do this, the input list/markers must be the output list/markers of the previous validation run and the retry_failed_portals argument must be set to True. The function will then rerun the validation, but will retry only the portals for which the

validation failed or the suspected API did not work previously.

When calling the function, three arguments are required and two are optional:

- input_list (string): the path of the CSV input file containing URLs - must be a file created previously by add_prefixes or validate_list - if retry_failed_portals is True, must be a file created previously by validate_list
- output_list (string): the path of the CSV output file to be exported, containing validated URLs
- output_markers (string): the path of the JSON output file to be exported, containing portals and their detected validation markers
- input_markers (string, optional): the path of the JSON input file containing portals and their detected validation markers - must be a file created previously by validate_list
- retry_failed_portals (Boolean, optional): whether or not to retry the portals for which the validation failed or the suspected API did not work in a previous run - defaults to False

Here is how we called the function for the first run:

```
validate_list(
    input_list = "data/4_prefixed_portals.csv",
    output_list = "data/5_validated_portals.csv",
    output_markers = "data/5_validated_sites.json")
```

Listing 12: Portal handler: validate list

And here is how we called the function to retry the portals that failed in the first run:

```
validate_list(
    input_list = "data/5_validated_portals.csv",
    output_list = "data/5_validated_portals_retry.csv",
    output_markers = "data/5_validated_sites_retry.json",
    input_markers = "data/5_validated_sites.json",
    retry_failed_portals = True)
```

Listing 13: Portal handler: validate list, retry failed portals

A.3.7 Analyze the validated list

Once the validation is done, the function analyze_list allows analyzing, presenting and saving the most important information about a validated portal list. Results are currently stored in the data/validation_statistics.csv file, which is hard-coded in the function definition so that the statistics of all validation runs are collected in one file.

In the validation statistics, the term "suspected" (e.g. in the column "ckan_suspected") is deliberately used instead of "markers". This is because portals added via the add_api_endpoints function will be counted as "suspected" for their respective portal software and thus, the number of "suspected" portals is not equivalent to the number of sites for which HTML markers were found.

When calling the function, one argument is required and two are optional:

- validated_portals_file (string): the path of the CSV input file containing validated portal URLs - must be a file created previously by validate_list
- show (Boolean, optional): whether or not to display the relevant DataFrames and results - defaults to True
- export (Boolean, optional): whether or not to append the results to the statistics CSV file - defaults to False

Here is how we called the function for the first run:

```
analyze_list(
    validated_portals_file =
    ↪ "data/5_validated_portals_retry.csv",
    show = True,
    export = True)
```

Listing 14: Portal handler: analyze list

A.3.8 Extract portals with working APIs

In the last step of the portal handler pipeline, the function extract_working_apis extracts the essential data from the validated portal list, keeps only the portals with working APIs, performs a final deduplication and exports the final list that will be used for portal crawling later.

When calling the function, two arguments are required:

- validated_portals_file (string): the path of the CSV input file containing validated portal URLs - must be a file created previously by validate_list
- output_file (string): the path of the CSV file to be exported, containing the final list of portal APIs

Here is how we called the function for the first run:

```
extract_working_apis(
    validated_portals_file =
    ↪  "data/5_validated_portals_retry.csv",
    output_file = "data/portals.csv")
```

Listing 15: Portal handler: extract working APIs

A.3.9 Check custom URL lists

Just like we already deployed our portal handler to check URLs submitted by colleagues, any arbitrary list of URLs can be run through and validated. If you want to combine your custom URLs with our 4 sources, edit the create_list function and add code that loads the URLs and appends them to the initial_portals DataFrame, then proceed with the other functions as usual. If you only want to check the custom URLs, save them in a CSV file that has a single column named "url" and skip the create_list function, instead pass the CSV file to the remove_duplicates function as input, then continue. When doing such a custom check, all presented rules and guidelines still apply and all functionality, for example adding manually validated endpoints that skip the first validation step, is available.

A.4 Portal crawling

Path: data_portal_tracker/portal_crawler.(ipynb|py)

Our **portal crawler** is the next major component of the Data Portal Tracker. It contains four crawling functions, each for a specific portal API software:

- **Opendatasoft API v1.0** (deprecated, only for completeness)
- **Opendatasoft API v2.1** (latest version)
- **CKAN API v2.x** (wide range, including latest version)
- **Socrata API v1.0** (only version of metadata API)

Which specific APIs the stated numbers refer to is explained in the thesis, however, please note that some of the crawling functions might partially use APIs that are different to the API to which the respective number belongs. For example, within the same portal API software, downloading datasets might sometimes be handled by a different API / service than displaying metadata.

Every function of the portal crawler takes the final portal list from the portal handler, loops through the corpus of every data portal and adds each dataset to the ODArchiver along with its metadata and a dataset/metadata mapping. In case of an exception, the last activity is retried up to three times before skipping the current loop iteration. Errors and, where applicable, also the dataset for which they occurred are saved for troubleshooting. Some crawling ideas and logic were taken from the unfinished Portal Watch API.

If you only want to count the numbers of datasets/resources per portal without crawling and indexing all datasets/resources, run the functions crawl_opendatasoft_v2, crawl_ckan and crawl_socrata after commenting out the code that handles the datasets and adds them to the Archiver:

```
1 # Calling the Archiver connector to insert data into the
  ↪ Archiver
2 archiver.handle_dataset(dataset_url, metadata_url, source_url,
  ↪ log_file_success, log_file_fail)
```

Listing 16: Portal crawler: code to comment out for counting datasets

A.4.1 Crawl Opendatasoft API v1.0

To crawl all portals on an input list that support the Opendatasoft API v1.0, you can use the function crawl_opendatasoft_v1. However, it was only created for completeness and to perform some comparisons with the Opendatasoft API v2.0 / v2.1 - Opendatasoft API v1.0 is now **deprecated** and we recommend using the function for v2.1 below!

As the Opendatasoft API v1.0 supports pagination, the function iterates over the metadata and datasets in batches of 800 until there are none left. For each set of dataset URL, metadata URL and source URL, it calls the ODArchiver connector to add the dataset, metadata and their mapping to the ODArchiver. Dataset URLs are built by assuming the availability of the CSV export format for all datasets, as explained in chapter 4.3 of the thesis. Optionally, if the portal is the Opendatasoft data hub[19], the source URL can be built differently than for all other portals by taking the URL of the original data source instead of the URL of the page that presents the dataset on the Opendatasoft data hub.

The two required arguments are the same as in all crawling functions:

- portal_list (string): the path of the CSV input file containing the final portal list - must be a file created previously by extract_working_apis in the portal handler
- statistics_file (string): the path of the CSV file to be created or extended, containing the statistics for the crawled portals

Here is how we called the function (during testing only):

```
crawl_opendatasoft_v1(
    portal_list = portal_list,
    statistics_file =
    ↪ "data/portal_statistics_opendatasoft_test.csv")
```

Listing 17: Portal crawler: crawl Opendatasoft v1

A.4.2 Crawl Opendatasoft API v2.1

All portals on an input list that support the Opendatasoft API v2.1 can be crawled with the function crawl_opendatasoft_v2. Even though API v2.0

[19] https://data.opendatasoft.com

also exists and the API responses of v2.1 differ slightly from v2.0, there is no need for a separate v2.0 function since every portal in our list that supports v2.0 also supports v2.1.

The logic of the function is similar to that of the v1.0 function, with two main differences. Firstly, since portals using API v2.1 offer a JSON metadata catalog of all datasets, the catalog is requested for each portal instead of using pagination. Secondly, the optional code from the v1.0 function is not present here, but there is another optional code section that checks the available export formats of each dataset and that was used to determine that CSV is generally available - which is why CSV is now assumed for building the dataset URL and the code is commented out. Per dataset, the function takes roughly 2 seconds, which means it can work through 30 datasets per minute or 1800 datasets per hour.

The two required arguments are the same as in all crawling functions:

- portal_list (string): the path of the CSV input file containing the final portal list - must be a file created previously by extract_working_apis in the portal handler
- statistics_file (string): the path of the CSV file to be created or extended, containing the statistics for the crawled portals

Here is how we called the function for the first run:

```
crawl_opendatasoft_v2(
    portal_list = portal_list,
    statistics_file =
     ↪ "data/portal_statistics_opendatasoft.csv")
```

Listing 18: Portal crawler: crawl Opendatasoft v2

A.4.3 Crawl CKAN API v2.x

The function crawl_ckan crawls all portals on the input list that support the CKAN API v2.x and has been tested with portals using a wide range of CKAN versions from v2.0 to v2.10.

Similar to the Opendatasoft API v1.0 function, pagination is used to loop through all datasets. However, on CKAN portals, one dataset/package can contain multiple resources. Therefore, unlike all other crawling functions, there is a nested loop for each dataset that iterates over all resources

of a dataset. For each set of resource URL, (dataset) metadata URL and (dataset) source URL, the function calls the ODArchiver connector to add the resource, metadata and their mapping to the ODArchiver.

The two required arguments are the same as in all crawling functions:

- portal_list (string): the path of the CSV input file containing the final portal list - must be a file created previously by extract_working_apis in the portal handler
- statistics_file (string): the path of the CSV file to be created or extended, containing the statistics for the crawled portals

Here is how we called the function for the first run:

```
crawl_ckan(
    portal_list = portal_list,
    statistics_file = "data/portal_statistics_ckan.csv")
```

Listing 19: Portal crawler: crawl CKAN

A.4.4 Crawl Socrata API v1.0

The function crawl_socrata crawls all portals on the input list that support the Socrata API v1.0.

Using pagination, the function loops through all datasets of each portal and checks whether the dataset type is one of two supported asset types for which our function can build dataset URLs: "dataset" or "file". If one of these types is found, the dataset and metadata are sent to the ODArchiver for indexing and mapping, otherwise the dataset is skipped. The asset types "chart", "datalens" and "filter" might work with the same method as the "dataset" type, but further testing is required to ensure that this approach is valid. We added a comment to the function that contains this information and a replacement for the current if-statement which can be used if testing shows positive results. For the type "map", future support is not likely because maps are purely front-end visualizations that use data from entities of the type "dataset", which are already supported and crawled.

The two required arguments are the same as in all crawling functions:

- portal_list (string): the path of the CSV input file containing the final portal list - must be a file created previously by extract_working_apis in the portal handler

- statistics_file (string): the path of the CSV file to be created or extended, containing the statistics for the crawled portals

Here is how we called the function for the first run:

```
crawl_socrata(
    portal_list = portal_list,
    statistics_file = "data/portal_statistics_socrata.csv")
```

Listing 20: Portal crawler: crawl Socrata

A.5 ODArchiver connection

Path: data_portal_tracker/archiver_connector.(ipynb|py)

Our ArchiverConnector contains methods which connect to the ODArchiver API (using HTTP requests) and the ODArchiver database (via MongoDB queries). In the subsections below, we describe how to interact with this class and its methods. Currently, the crawling script calls the ArchiverConnector's class constructor and all crawling functions call its handle_dataset method which then calls all other class methods.

For manual interactions with the ODArchiver's MongoDB, here is some information about the database schema:

- datasets: essential information for crawler to work - two unique identifiers, _id and id, an array of versions and the three objects meta, url and crawl_info
- datasets.files: information about the individual versions of a dataset, referenced by their IDs from the versions array of the respective datasets document
- datasets.chunks: actual data of the files, stored as chunked, Base64-encoded binaries - the files_id field references the _id field in the files collection.
- datasets.mappings: newly added as part of the Data Portal Tracker, contains the mappings of datasets and metadata - the dataset_id, metadata_id and added fields store the IDs of the dataset and metadata as well as the timestamp of the mapping creation
- hosts: necessary information for the locking mechanism and host politeness to work properly, for example the currentlyCrawled field
- sources: referenced by the source array field in the meta object of the datasets collection - one dataset can have multiple sources and one source can be referenced by multiple datasets

When performing queries in the ODArchiver MongoDB without using the methods we provided, keep in mind that some fields, for example _id in the datasets collection, use the type ObjectID rather than string.

A.5.1 Instantiate the class

ArchiverConnector is instantiated by its __init__ method which attempts to establish a connection to the MongoDB and prints information on the success or failure. By passing an argument, the database to connect to can be chosen: If "production" is passed, the three nodes of the Kubernetes cluster that the ODArchiver's MongoDB runs on are tried out one after another. For testing, a local database can be used by passing "local" and ensuring that the local connection string and database name in the .env file match a local MongoDB that has the same database structure and a collection named datasets.mappings.

One argument is required:

- mode (string): which MongoDB to connect to - must be "local" or "production"

Here are the two ways of calling the class constructor:

```
1 archiver = ArchiverConnector(mode = "production")
2 # archiver = ArchiverConnector(mode = "local")
```

Listing 21: ODArchiver connector: call class constructor

A.5.2 Get dataset information via API

The method api_get_dataset takes a dataset URL, encodes it and performs an API request to check if the dataset is already indexed by the ODArchiver. Keep in mind that all ODArchiver API methods refer to "datasets", but now also apply to metadata because of the extensions we made to the system.

One argument is required:

- dataset_url (string): the URL of the dataset

A dictionary containing four items is returned:

- request_success: whether the request was successful
- dataset_found: whether the dataset was found
- dataset_id: the ID of the dataset in the ODArchiver or None

- message: success message or failure message with details about the error

Here is how we called the method for testing:

```
archiver = ArchiverConnector(mode = "production")
dataset_url = "http://data.cookcountyil.gov/download/ikxe-tdm7"

archiver.api_get_dataset(dataset_url = dataset_url)
```

Listing 22: ODArchiver connector: get dataset information from API

A.5.3 Add dataset via API

The method api_add_dataset takes a dataset URL and a source URL and performs an API request to add the dataset to the ODArchiver. Keep in mind that all ODArchiver API methods refer to "datasets", but now also apply to metadata because of the extensions we made to the system.

Two arguments are required:

- dataset_url (string): the URL of the dataset
- source_url (string): the URL of the dataset's source

A dictionary containing three items is returned:

- request_success: whether the request was successful
- dataset_inserted: whether the dataset was inserted
- message: success message or failure message with details about the error

Here is how we called the method for testing:

```
archiver = ArchiverConnector(mode = "production")
dataset_url = "http://data.cookcountyil.gov/download/ikxe-tdm7"
source_url = "http://data.cookcountyil.gov/d/ikxe-tdm7"

archiver.api_add_dataset(
    dataset_url = dataset_url,
    source_url = source_url)
```

Listing 23: ODArchiver connector: add dataset via API

A.5.4 Get mapping via database

The method mongodb_get_mapping checks if there is an existing mapping between a dataset and its metadata in the "datasets.mappings" collection of the MongoDB.

Two arguments are required:

- dataset_id (string): the ID of the dataset in the ODArchiver
- metadata_id (string): the ID of the metadata in the ODArchiver

A dictionary containing five items is returned:

- query_success: whether the query was successful
- dataset_found: whether the dataset was found in any mapping
- metadata_found: whether the metadata was found in any mapping
- mapping_found: whether a mapping between the dataset and the metadata was found
- message: success message or failure message with details about the error

Here is how we called the method for testing:

```
archiver = ArchiverConnector(mode = "production")
dataset_id = "64db96b381165e001229e325"
metadata_id = "64db96b382d36f00137ff647"

archiver.mongodb_get_mapping(
    dataset_id = dataset_id,
    metadata_id = metadata_id)
```

Listing 24: ODArchiver connector: get mapping via database

A.5.5 Add mapping via database

The method mongodb_add_mapping adds a mapping entry for a given dataset ID and metadata ID to the "datasets.mappings" collection of the MongoDB.

Two arguments are required:

- dataset_id (string): the ID of the dataset in the ODArchiver
- metadata_id (string): the ID of the metadata in the ODArchiver

A dictionary containing three items is returned:

- inserted: whether the mapping was inserted
- mapping_id: the ID of the mapping document or None
- message: success message or failure message with details about the error

Here is how we called the method for testing:

```
archiver = ArchiverConnector(mode = "production")
dataset_id = "64db96b381165e001229e325"
metadata_id = "64db96b382d36f00137ff647"

archiver.mongodb_add_mapping(
    dataset_id = dataset_id,
    metadata_id = metadata_id)
```

Listing 25: ODArchiver connector: add mapping via database

A.5.6 Handle dataset

The method handle_dataset checks if a dataset and its metadata are both already indexed by the ODArchiver and have a mapping that describes their relation. Any missing indexing or mapping is added.

Five arguments are required:

- dataset_url (string): the URL of the dataset
- metadata_url (string): the URL of the dataset's metadata

- source_url (string): the URL of the dataset's source
- log_file_success (string): the path of a CSV file logging successfully handled datasets
- log_file_fail (string): the path of a CSV file logging datasets for which an exception occurred

A dictionary containing seven items is returned:

- success: whether the process was successfully completed
- dataset_added: whether the dataset was inserted via the API
- metadata_added: whether the metadata was inserted via the API
- mapping_added: whether a mapping between the dataset and the metadata was added via the MongoDB
- dataset_id: the ID of the dataset in the ODArchiver or None
- metadata_id: the ID of the metadata in the ODArchiver or None
- message: success message or failure message with details about the error

Here is how we called the method for testing:

```
archiver = ArchiverConnector(mode = "production")
dataset_url = "http://data.cookcountyil.gov/download/ikxe-tdm7"
metadata_url =
↪  "http://data.cookcountyil.gov/api/views/metadata/v1/ikxe-tdm7"
source_url = "http://data.cookcountyil.gov/d/ikxe-tdm7"
log_file_success = "logs/handle_dataset_TEST_success.csv"
log_file_fail = "logs/handle_dataset_TEST_fail.csv"

archiver.handle_dataset(
    dataset_url = dataset_url,
    metadata_url = metadata_url,
    source_url = source_url,
    log_file_success = log_file_success,
    log_file_fail = log_file_fail)
```

Listing 26: ODArchiver connector: handle dataset

A.6 Helper functions

Path: data_portal_tracker/helpers.py

We have created multiple functions to support URL processing tasks in the portal handler and portal crawler scripts. These utilities were designed to be reused in different components of our system, thus we collected them in a separate script only containing helpers, which enables intuitive and simple importing of the functions wherever they are needed.

A.6.1 Check website activity

The first helper function check_url requests a well-formed input URL that has a protocol prefix and returns information about the response. This function is currently called by check_protocol, another helper function.

One argument is required:

- url (string): the URL to be requested - must include a protocol prefix (http:// or https://)

A dictionary containing three items is returned:

- request_success: whether the request was successful
- response_code: the HTTP response code
- message: success message or failure message with details about the error

A.6.2 Check website protocol

Next, check_protocol takes an input URL with or without protocol prefix and finds out which protocol is working for the URL. It requests a URL with HTTPS and, if required, falls back to HTTP and finally returns the "best" working variant of the URL in this order: HTTPS, HTTP, URL without prefix. This function is currently called by the function add_prefixes in the portal handler.

One argument is required, one is optional:

- url (string): the URL for which the protocol should be checked - can be with or without HTTP(S) prefix

- show_details (Boolean, optional): whether or not to print details about the requests - defaults to True

One string is returned:

- str: the working URL with protocol prefix (HTTPS > HTTP) or non-working URL without protocol prefix

A.6.3 Remove double slashes

Finally, remove_double_slashes removes redundant forward slashes by replacing a double forward slash with a single forward slash in any part of a HTTP(S) URL except for the protocol prefix. In the context of the portal_crawler script, only apply this function to the substring of a URL related to the API, so before adding a dataset-specific ID or something similar! The function is currently called in all crawling functions of the portal crawler.

One argument is required:

- url (string): the URL to be modified which may contain double forward slashes

One string is returned:

- str: the modified URL with only single forward slashes

A.7 Experiments

Path: data_portal_tracker/experiments.ipynb

In this section, we will briefly describe experiments that we carried out to justify implementation decisions, to use our portal validation on other data and to prepare future work. More details, the code and all results can be found in the data_portal_tracker/experiments.ipynb notebook.

A.7.1 Validating "www." URLs with and without "www."

This experiment supported the decision-making process regarding the deduplication of URLs in the portal handler. Since a minority of URLs were appearing in the list twice, once with and once without the "www." prefix, the question was whether this prefix could be removed in the early deduplication step in the portal handler or if this would cause problems, e.g. a large number of sites not responding to requests anymore.

Based on the results of the experiment, which showed that 3 CKAN portals, 10 Opendatasoft portals and 4 Socrata portals would be lost when removing "www." early on, we have decided not to remove the "www." prefix in the early deduplication step. Taking into account the small share of the described duplicates among all URLs, the benefits derived from this deduplication (reducing the number of HTTP requests during validation, avoiding any duplicate sites in the list) are not outweighing the disadvantages ("losing" 10-20 Open Data portals with working APIs), especially since one main goal is to collect as many Open Data portals as possible and there are some interesting portals in the list of portals broken by the prefix removal, like the Open Data portals of Bahrain, Wallonia, Corsica and the City of Dallas.

Instead, any remaining duplicates that appear with and without "www." are removed from the list of portals with working APIs near the end of the pipeline, after the validation step. See the portal handler for details.

An alternative solution would have been to request every URL with and without the "www." prefix, similar to the already implemented function that performs an HTTPS request and, if necessary, an HTTP request to determine the best available protocol. However, this would have lead to multiple additional requests for many of the approximately 5000 sites as multiple combinations of HTTP or HTTPS and WWW or no WWW would have had to be tried out and would have vastly exceeded the reduction in requests from removing just over 100 duplicates.

A.7.2 Validating Opendatasoft file export formats

This experiment, which informed our decision on always assuming the availability of CSV, was already described in detail in section 4.3 of the thesis.

A.7.3 Validating railway and university portals

We deployed our portal handler to validate different Open Data portals of railway companies and universities at the request of interested colleagues.

Our validation of the given railway portals showed 2 working CKAN portals and 3 working Opendatasoft portals which are now also included in our main portal list. The portal of Deutsche Bahn (DB) is supposed to be based on CKAN, but the endpoints do not work, while the portal of the Austrian Federal Railways (ÖBB) is not using CKAN, Opendatasoft or Socrata. For Prorail, there is a portal that is based on ArcGIS and thus out of scope currently, but there is also some Prorail data in the Dutch government's Open Data portal[20] which has a working CKAN API but contains much more than just railway data.

Only two of the given educational organizations have a working API that is based on one of the portal software options we support (CKAN, Opendatasoft, Socrata). Of those two, one is the US Department of Education, the other is California State University and both of them use CKAN.

A.7.4 Validating the ODPW list

This experiment, in which we validated the portals on the ODPW list, provided new data for our analysis of the list's evolution from 2016 to 2023. The full methodology can be found in section 4.4 of the thesis and the results are presented in section 5.2.

A.7.5 Checking false positives of marker validation

On some sites, validation markers can be found in the first part of the validation step, but no working API is located subsequently. Given the chosen approach of only testing the API functionality on sites for which the validation marker search has been successful, these cases, which could be described as false positives, are worth investigating. We wrote some very short and simple code to display such cases that can be used as a starting point for future work in this area and is also located in the experiments notebook.

[20] https://data.overheid.nl/data

A.7.6 Checking DCAT extension on CKAN portals

CKAN offers an extension that enables the retrieval of metadata using the Data Catalog Vocabulary (DCAT). The code provided in the experiments notebook shows how to check the availability of this extension as well as the TTL / RDF catalog for all CKAN portals on the list. More details can be found in section 4.4 of the thesis and the results are shown in section 5.2.

A.8 Extending the system

Since users of the Data Portal Tracker may want to add support for further portal software other than CKAN, Opendatasoft and Socrata, this section details the required steps to make the relevant modifications.

1. (Optional:) Firstly, when including a different portal solution, it may make sense to consider this change already in the very first part of the pipeline, the search engine portal discovery. Searches are currently being carried out via the Crawley command line tool, see crawley-lite/README.md and the relevant sections above in the thesis and documentation. The search terms used are manually entered - some suggestions and previously used examples can be found in the values of the "search" keys in crawley-lite/config.json. To improve the end results and get more portals with a validated, working API, search terms should include relevant keywords like "Open Data Portal + [Name of API software]". However, the current list of potential portals is already very long (more than 5500 websites), so that most large portals on the web should already be included, they just couldn't be validated successfully yet as they do not use CKAN, Opendatasoft or Socrata. Therefore, this step is optional.

2. To adapt the first validation step, edit the crawley-lite/config.json file and add a top-level JSON key with the name of the software (this will be used in multiple locations throughout Data Portal Tracker, thus being consistent is recommended) whose value is an object containing the keys "search" and "validate", each with an array of strings as their value. Identify validation markers (HTML elements that can be frequently found in the HTML code of portals using the relevant software) by using the developer tools of any web browser and searching for keywords like the name or an abbreviation of the software. These markers must go in the array belonging to "validate", while "search" does not need any values. Listing 27 below shows an extract of config.json with these mentioned additions at the end. Note: values for the three "search" keys were omitted in the listing because they are currently not used by any script.

```
{
  "CKAN": {
    "search": [
    ],
    "validate": [
```

```
      "img/ckan.svg",
      "<a href=\"http://docs.ckan.org/en/2.8/api/\">CKAN API</a>",
      "<a href=\"http://docs.ckan.org/en/2.9/api/\">CKAN API</a>",
      ">CKAN API</a>",
      "<form id=\"ckan-dataset-search\"",
      "<a id=\"ckan_de\"",
      "<a id=\"ckan_en\"",
      "header_od_ckan.en",
      "/ckan/organization",
      "ckan-footer-logo",
      "wpckan_dataset",
      "id='ckan_base-js'",
      "<meta name=\"generator\" content=\"ckan",
      "ckan.ico",
      "href=\"/ckan/dataset",
      "<span>Powered by</span>",
      "alt=\"CKAN\"",
      "alt=\"CKAN logo\"",
      "href=\"/fanstatic/ckanext-harvest",
      "href=\"/data/fanstatic/ckanext-scheming",
      "href=\"/ckan/fanstatic/ckanext-geoview",
      "href=\"/ckan/fanstatic/ckanext-harvest"
    ]
  },
  "OpenDataSoft": {
    "search": [
    ],
    "validate": [
      "BRAND_HOSTNAME: \"opendatasoft.com\"",
      "ods.core.config",
      "ods.minimal",
      "ods.core.config",
      "ods.core",
      "ods.core.form.directives"
    ]
  },
  "Socrata": {
    "search": [
    ],
```

```
    "validate": [
      "<!-- Start of socrata Zendesk Widget script -->",
      "var socrata",
      "window.socrata",
      "<!-- \n            Powered by Socrata",
      "<!-- Powered by Socrata",
      "<!--Powered by Socrata",
      "<!--\n  Powered by Socrata",
      "www.socrata.com\n  -->"
    ]
  },
  "Name of new API software": {
    "search": [
      "No need to add anything here - 'search' values are
↪   currently unused!"
    ],
    "validate": [
      "Some HTML element",
      "Another HTML element",
      "A third HTML element"
    ]
  }
}
```

Listing 27: How to extend the crawley-lite/config.json file

3. In the second validation step, add code that validates the functionality of the new software's API to the validate_list function of the data_portal_tracker/portal_handler.(ipynb|py) script. The relevant section of the function is shown in listing 28 below (code shortened, see the script for all details). On the same indentation level as the three comments "# Checking portals with (CKAN|Socrata|Opendatasoft) markers" and the subsequent code blocks started by if-statements matching the "suspected_api" field of the prefixed_portals DataFrame, add another comment and if-statement for the new software. Research the new API to find a method that returns general information about the API like the supported methods, any extensions, the status, the current API version and any older supported versions. Save the version information to the "api_version" field and the information whether the API could be validated and is working (as determined by the request giving the expected result) to the "api_working" field of the prefixed_portals

DataFrame. If there is no suitable API method, perform any general request to the API (for example a "help" method) and, using a try/except block, try to access a known field of the response - if there is no error, mark the API as working, otherwise as not working. The exact code will vary depending on the API and will require some experimentation.

```
# Verifying that the detected API is available and working
if prefixed_portals.loc[index, "suspected_api"] is not None and
 ↪  prefixed_portals.loc[index, "suspected_api"] != "Unknown":

    # Checking portals with CKAN markers
    if prefixed_portals.loc[index, "suspected_api"] == "CKAN":
        # Resetting version variable
        ckan_version = None

        try:
            api_url = base_url + "/api/3/action/package_search"
            response = requests.get(api_url, timeout = 15)
            if json.loads(response.text)["success"] == True:
                print("CKAN API working")
                prefixed_portals.loc[index, "api_working"] =
                 ↪  True
                # Checking the API version
                try:
                    api_version_url = base_url +
                     ↪  "/api/3/action/status_show"
                    response = requests.get(api_version_url,
                     ↪  timeout = 15)
                    ckan_version =
                     ↪  json.loads(response.text)["result"]["ckan_version"]
                    prefixed_portals.loc[index, "api_version"]
                     ↪  = ckan_version
                except Exception as e:
                    prefixed_portals.loc[index, "api_version"]
                     ↪  = "Unknown"
                    log(e)
        except Exception as e:
            print("CKAN API not working")
            prefixed_portals.loc[index, "api_working"] = False
            log(e)
```

```
28
29      # Checking portals with Socrata markers
30      elif prefixed_portals.loc[index, "suspected_api"] ==
        ↪  "Socrata":
31          [...]
32
33      # Checking portals with Opendatasoft markers
34      elif prefixed_portals.loc[index, "suspected_api"] ==
        ↪  "OpenDataSoft":
35          [...]
36
37      # Checking portals with [Name of new software] markers
38      elif prefixed_portals.loc[index, "suspected_api"] == "[Name
        ↪  of new software]":
39          [...]
```

Listing 28: How to extend the validate_list function

4. For the analysis step in which statistics about each validated portal list are displayed and saved, edit and extend the function analyze_list of the data_portal_tracker/portal_handler.(ipynb|py) script. Add two new columns to the code creating the "statistics" DataFrame at the beginning: one for the portals suspected to be using the new software (based on the results of marker-based validation) and one for the portals confirmed to be using the new software (based on the results of the API validation). For both of these subsets of portals, add code that counts, displays and saves them. As a reference and template, take the code for CKAN (shown in listing 29 below) or Opendatasoft or Socrata (omitted below), just with "CKAN/Opendatasoft/Socrata" replaced by the name of the new software in every string and variable name.

```
1  # Creating a DataFrame for the statistics
2  statistics = pd.DataFrame(columns = ["file", "total", "active",
   ↪  "inactive", "validated", "unvalidated",
   ↪  "subpage_endpoints", "no_markers", "ckan_suspected",
   ↪  "ckan_working", "opendatasoft_suspected",
   ↪  "opendatasoft_working", "socrata_suspected",
   ↪  "socrata_working", "name_of_new_software_suspected",
   ↪  "name_of_new_software_working", "timestamp"])
3  [...]
```

```
4
5  # Suspected CKAN portals (validation markers found or API
   ↪  manually checked)
6  ckan_markers_portals =
   ↪  validated_portals[validated_portals["suspected_api"] ==
   ↪  "CKAN"]
7  ckan_markers_portals.columns.name = "Suspected CKAN portals"
8  statistics.loc[0, "ckan_suspected"] = len(ckan_markers_portals)
9  if show is True:
10     display(ckan_markers_portals)
11
12 # Portals with working CKAN API
13 ckan_working_api_portals =
   ↪  ckan_markers_portals[ckan_markers_portals["api_working"] ==
   ↪  True]
14 ckan_working_api_portals.columns.name = "Portals with working
   ↪  CKAN API"
15 statistics.loc[0, "ckan_working"] =
   ↪  len(ckan_working_api_portals)
16 if show is True:
17     display(ckan_working_api_portals)
18
19 # Suspected Opendatasoft portals (validation markers found or
   ↪  API manually checked)
20 [...]
21
22 # Portals with working Opendatasoft API
23 [...]
24
25 # Suspected Socrata portals (validation markers found or API
   ↪  manually checked)
26 [...]
27
28 # Portals with working Socrata API
29 [...]
30
31 # Suspected [Name of new software] portals (validation markers
   ↪  found or API manually checked)
32 [...]
33
34 # Portals with working [Name of new software] API
```

```
35 [...]
```

Listing 29: How to extend the analyze_list function

5. Extend the data_portal_tracker/portal_crawler.(ipynb|py) script by adding a function that crawls all dataset URLs, metadata URLs and source URLs of portals based on the new software and model it after the existing functions. This is likely the most time-intensive step, as the API structure may deviate from those of the CKAN, Opendatasoft and Socrata APIs. Try to find out as much as possible about the new API and take the most suitable of the four finished functions as a template:

   - If datasets contain resources and the metadata API supports or requires pagination: crawl_ckan function.
   - If datasets are resources and the metadata API supports or requires pagination: crawl_socrata or crawl_opendatasoft_v1 function.
   - If datasets are resources and the metadata API offers a JSON metadata catalog of all datasets: crawl_opendatasoft_v2 function.
   - In any other case, a combination of the relevant code sections might be helpful.

   Then, when adapting the chosen code for the new platform, it is advisable to first comment out most lines except for the very start and work your way forward line by line. In any case, when modifying the code and possibly altering loops or removing sleep calls, definitely initially comment out any HTTP requests and calls of the handle_dataset function imported from data_portal_tracker/archiver_connector.py to prevent accidental denial-of-service attacks on WU's or external servers - see listing 30.

```
1 response = json.loads(requests.get(api_request_url).text)
2 [...]
3             archiver.handle_dataset(resource_url, metadata_url,
              ↪ source_url, log_file_success, log_file_fail)
4             [...]
```

Listing 30: Code to comment out when testing an adapted crawling script