# Mapping Research Software Landscapes through Exploratory Studies of GitHub Data

Utrecht University

A MASTER THESIS PRESENTED
BY
KEVEN QUACH
TO
THE DEPARTMENT OF INFORMATION AND COMPUTING SCIENCES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE
IN THE SUBJECT OF
BUSINESS INFORMATICS

UTRECHT UNIVERSITY
OCTOBER 2022

FIRST SUPERVISOR: ANNA-LENA LAMPRECHT
SECOND SUPERVISOR: FERNANDO CASTOR DE LIMA FILHO
DAILY SUPERVISOR: JONATHAN DE BRUIN

Thesis advisor: Anna-Lena Lamprecht Keven Quach

# Mapping Research Software Landscapes through Exploratory Studies of GitHub Data

## Abstract

Research software enables data processing and plays a vital role in academia and industry. As such, it is essential to have findable, accessible, interoperable, and reusable (FAIR) research software. However, what precisely the landscape of research software looks like is unknown. Thus, we would like to understand the research software landscape better and utilize this information to infer actionable recommendations for the Research Software Engineer (RSE) practice. This study provides insights into the research software landscape at Utrecht University through an exploratory analysis while also considering the different scientific domains. We achieve this by collecting GitHub data and analyzing repository FAIRness and characteristics through heatmaps, histograms, statistical tables, and tests. Our method retrieved 176 users with 1521 repositories, of which 823 are considered research software. Others can adopt the proposed method to gain insights into their specific organization, as it is designed to be reproducible and reusable. The analysis showed significant differences between faculty characteristics and how to support the application of FAIR variables. Among other things, our results showed that Geosciences have the highest percentage of unlicensed repositories with 57%. Also, Social Sciences are an outlier in language usage, as they are the only faculty to primarily use R. Other faculties primarily use Python. A first classification model is developed that achieves 70% accuracy in identifying research software that can be used for future labelling tasks. We conclude that our labelled GitHub dataset allows us to infer actionable recommendations on RSE practice.

# Contents

# Acknowledgments

I would like to thank Anna-Lena Lamprecht, my first supervisor, and Jonathan de Bruin, my daily supervisor, for working together on the SWORDS project and providing me with the opportunity to conduct this study based on this previous work. It has been an invaluable experience for me. Additionally, I want to thank Fernando Castor for offering his advice as a second supervisor, as well as the FAIR Data and Software track as they have been very supportive during my time as a research assistant in the track.

# 1 Introduction

Data and research software play a vital role in modern research, enabling scientific breakthroughs like capturing a black hole as an image [34, 75], storing massive datasets on distributed systems [96], or the open-source publication of a system for large-scale machine learning, which allows everyone to solve computer vision tasks, among many other things [27]. However, not much is known about the landscape of research software, which researchers and support staff create, and scientific results may not be reproducible or valid [32]. Relevant information is scattered across different platforms like GitHub, PapersWithCode.com, employee pages and more. Some entities require researchers to follow strict publishing guidelines to solve these issues, and some initiatives try to gather research software in a system like the research software directory [98], which was developed by the Netherlands eScience Center and can be implemented for different scopes. However, a deeper analysis of such data to infer actionable recommendations for Research Software Engineer (RSE) practice has not been conducted yet.

This research project aims to develop a method for conducting exploratory data analyses of GitHub data, which is transferable to other organizations and thus contributes to the scientific community. GitHub is the most popular development platform in research and is the go-to reference for mining open source repositories [48]. This method is then applied to data from Utrecht University (UU) only to maintain feasibility but can be applied to other use cases such as comparisons between universities or other organizations. The data used for the project is collected using the Scan and revieW of Open Research Data and Software at Utrecht University (SWORDS@UU) framework [49], which was initiated by Anna-Lena Lamprecht and Jonathan de Bruin to get insights into how UU researchers develop, manage and publish software.

## 1.1 Relevance

This research is highly relevant for both the academic and non-academic worlds of software engineering, while the main focus lies in academia. This research output can allow an organization to make data-based strategic decisions, for example, regarding software openness and sustainability. Support staff can use the insights to craft specialized trainings as knowledge gaps and needs might differ per faculty or department. This will also help improve researchers' assistance in achieving publications and research software that are more findable, accessible, interoperable, and reusable (FAIR). Therefore, the output of this work supports the application of FAIR principles for research software which helps to improve the research software landscape. The proposed method is also publicly available so that other organizations can adopt it for their own purposes.

In addition, the FAIR principles share some ideas with Free and/or Open Source Software (FOSS) objectives, which is also highly relevant for non-academic software engineering since much software is built upon other FOSS software [79]. An overwhelming majority of software source code originates from the non-academic world, namely industry and developer communities [92]. Thus, support for applying FAIR principles is also relevant for non-academic practice. In fact, there are many initiatives and organisations embedded within the industry that aim to develop FOSS software, such as the Eclipse Foundation, Open Manufacturing Platform, and Catena-X.

## 1.2 Research question

This study aims to answer the following research question:

- How can information about open source publications on GitHub be used to infer actionable recommendations for RSE practice to improve the research software landscape of an organization?

The main research question will be answered with the help of the following subquestions:

1. What is the current state of the art of FAIR principles?

2. How can FAIR principles be supplemented with additional variables?
3. Are there different characteristics for different subpopulations[1] in the data?
4. How can the application of FAIR variables for research software be supported?
5. How well can we identify research software with available data?

As stated in subquestion 2, the need for measurable variables becomes evident when one aims to better understand research software FAIRness in quantitative terms. There is currently only one study available that proposes a first FAIR-like framework to measure software quality, but it does not apply to data only from GitHub and focuses on a single domain [88]. Subquestions 2 and 4 should also be contemplated for the different subpopulations as this, for example, enables the crafting of the aforementioned specialized trainings. Answering these questions will help to gain a better understanding of a research software landscape, and what differences exist for the different subpopulations.

Subquestions 1 and 2 are answered through a literature review. Subquestion 2 will additionally be validated through a similarity analysis. The other subquestions will be answered through the exploratory data analysis and classification, which are explained in more detail in Section 3.2

## 1.3 Thesis outline

This study continues with the literature review and necessary background information in Chapter 2. In Chapter 3, the research method and planning of the study are presented. Chapter 4 shows the results of the analysis that are then discussed in Chapter 5, followed by the conclusion in Chapter 6.

[1] Subpopulations refer to grouping levels like faculties, departments and positions (e.g. PhD candidate, associate professor), while the population refers to all identified employees.

# 2 Background and state of the art

This Chapter explains the necessary background knowledge in Section 2.1 and comprises the literature review in the following Sections. The literature review includes the protocol in Section 2.2, FAIR principles in general and for research software in Section 2.3 and Section 2.4, and existing GitHub analyses of FAIRness and research domains in Section 2.5.

## 2.1 Technical background knowledge

*GitHub* is a development platform from which data for this study is collected. The platform allows developers and organizations to collaborate in open or private repositories based on the version control system Git [42]. Common ways for collaboration are creating *issues* or *pull requests*. Issues are used to track feedback, ideas, issues like bugs, feature requests and more. Pull requests allow users to commit any changes to the repository that a user would like to suggest. This is usually done by *forking* a repository, which essentially means cloning. Afterwards, the contributor *commits* and *pushes* changes on the forked repository and *merges* the changes to the original one via pull requests. It also offers the opportunity to follow people to get notified about their activity. GitHub also offers many other features for free, like *GitHub Actions*, which allows for integrated continuous integration and continuous delivery or continuous deployment (CI/CD) pipelines, and automatic detection of citation and license files, leading to machine-readable metadata. There is also a *REST API* [5] which allows us to extract information about users and repositories with all accompanying metadata that is available. Such REST APIs use the HTTP protocol to receive and modify data. To make things easier, we use ghapi [11] in the SWORDS@UU framework, a Python wrapper for the GitHub REST API.

In the remainder of the thesis, we will use the terms *metric* and *(FAIR) variable*. A variable is any kind of captured data like the owner of the repository or the repository id. A FAIR variable is related to concepts of

FAIRness, openness, and sustainability. Examples of FAIR variables are the availability of a license or citation information. A *metric* is any kind of numeric variable like the number of issues, forks or stargazers. A metric is always a variable and can also be a FAIR variable, but a (FAIR) variable does not always have to be a metric.

The FAIR for Research Software (FAIR4RS)-subgroup3 reviewed definitions of research software and provided their own definition, which will be used for the remainder of the study [60]:

> **Definition 1** *Research Software includes source code files, algorithms, scripts, computational workflows and executables that were created during the research process or for a research purpose. Software components (e.g., operating systems, libraries, dependencies, packages, scripts, etc.) that are used for research but were not created during or with a clear research intent should be considered software in research and not Research Software. This differentiation may vary between disciplines. The minimal requirement for achieving computational reproducibility is that all the computational components (Research Software, software used in research, documentation and hardware) used during the research are identified, described, and made accessible to the extent that is possible. [60]*

The creation of the definition has been a controversial discussion. During the analysis of determining whether a software is considered a research software, there were often conflicting opinions. Katz et al. [70] also stated that it is hard to distinguish between research and non-research initiatives due to an increasing focus on open source software in general.

An essential aspect of the research software definition is the variation between disciplines. What exactly is determined as research software depends on the agreement within communities and may also change over time. This allows for flexibility since software is highly complex and makes the definition applicable in the future as well [79]. Software can also have different granularity levels, making it difficult to determine and reference the appropriate level correctly [92]. However, these aspects also make it difficult to categorize research software properly. Since the categorization is flexible, labelled data from today might not be correctly labelled in the future.

## 2.2 Literature review protocol

The relevant literature includes FAIR principles in general, specifically regarding research software and the analysis of open source GitHub repositories. To that effect, the snowball method [105] was applied for identified key papers [72, 79, 104]. Both forward and backward snowballing were applied. Forward snowballing refers to following papers that have cited a specific paper, while backward snowballing refers to following the references in a specific paper. There are also reading lists containing relevant research published before and after February 2020 for the application of FAIR principles for software that were taken into account [25, 103]. In addition, Google scholar was used to identify additional papers with the queries *FAIR research software*, *FAIR research software GitHub*, and *GitHub analysis metadata open*. The snowball method was again applied to identified key papers [63].

## 2.3 FAIR principles in general

The FAIR principles were published in 2016 by Wilkinson et al. [104] to clarify the objectives of good data management and are concise, domain-independent and high-level. Good data management acts as a pre-condition for better knowledge discovery and innovation. The initiative initially started in a workshop in 2014 with many involved parties interested in data discovery and reuse. The four foundational FAIR principles aim to improve ***F**indability, **A**ccessibility, **I**nteroperability, and **R**eusability*. They are accompanied by guiding principles that further elaborate on each foundational principle. A reference can be found in Appendix A, which also includes proposed FAIR principles for research software.

An important aspect of the FAIR principles is that they should apply to all digital research objects like algorithms, tools, workflows, and research software. To allow for reproducibility and reusability of research, it is

necessary to make all relevant objects of the research process FAIR. There has also been an increase in general-purpose data repositories, with Zenodo [53] being one well-known and widely used example. In contrast to special-purpose repositories, general-purpose ones lead to less integrated data ecosystems that reinforce issues in discovery and reusability. Scientific data gathering, therefore, is a time-consuming process that can be improved. The goal is to include aspects of FAIRness in the quality assessment of digital research objects.

Wilkinson et al. stated that the FAIRification of digital research objects leads toward machine-actionability. Machine-readability becomes more and more important as humans increasingly rely on computational agents for discovery and integration. In the optimal case, such a machine can autonomously explore digital objects for different tasks: Identify object type, determine usefulness for a given context, determine usability considering license, accessibility or usage constraints, and take appropriate action. Nonetheless, the FAIR principles are aimed toward human-driven and machine-driven activities, which is different from similar initiatives. They also precede implementation and are technology-agnostic, allowing for high flexibility and applicability across domains. However, they are not a standard and primarily act as a guideline. The entry barrier for relevant operators is kept low by only minimally defining the FAIR guiding principles. The foundational principles are also independent and separable, though they are still related.

In 2017, a follow-up paper [85] by the original authors of the FAIR principles was released to clarify the intent and interpretation of the FAIR Principles. They emphasized again that FAIR principles focus on making research objects reusable to maximize their value without specifying technical requirements and that it is not a standard. Instead, they are permissive guidelines that can be used to develop flexible community standards [39, 85]. In addition, they stated that open science is steadily growing, and FAIR principles within this context have been embraced by a wide range of governments and funding bodies. What is still needed are changes to science reward and methodological practice, as well as increased support infrastructure. Resources in such infrastructures should be FAIR. For this purpose, community-acceptable rules of engagement should be developed

that are suitable for a given community. They also clarify that "unfair" does not exist. For example, if data is non-machine-readable, it does not become unfair. Rather, FAIRness is a continuous spectrum. Making associated metadata FAIR, in case the data may not be shared if it is for example personal data, already counts as full participation within the FAIR ecosystem. This also showcases that openness and FAIR are not the same. Data, especially health data, can often be on an individual level and thus resulting in privacy concerns which lead to mixed success regarding appeals for open data [39, 41]. This is of less concern for FAIR research software and one of the significant differences between data and software. They introduce the term re-useless data for datasets that are not available for reuse, which is the majority of datasets with 80% [85]. The first step to making such datasets FAIR is providing a Persistent Unique Identifier (PID), but that alone is insufficient. The next step would be adding FAIR metadata, followed by making the data FAIR. The last step would then be to link the data with other FAIR data to achieve the internet of FAIR data. What can be taken away from this is that improving FAIRness is a continuous effort, and it is not clear when exactly something is FAIR.

There has been research on how FAIR principles can be applied to other digital research objects such as computational workflows, training material, machine learning tools and models [55, 74, 106]. For machine learning specifically, Katz et al. [74] started a community-building process for applying FAIR concepts to that domain. The FAIR4RS-subgroup2 was tasked with exploring FAIR in contexts other than data and software [78]. Common recommendations based on their analysis include rich metadata (e.g. citation), recognition, proper documentation, unique identification of material (e.g. via PIDs), and getting community endorsement. We will see in Section 2.4 that these recommendations are also highly relevant and applicable to research software.

## 2.4 FAIR principles for Research Software

Since the original FAIR principles were published, several publications have pointed out that these principles do not directly translate to research software [61, 62, 79]. Thus, it is necessary to modify, extend, add and delete these principles to make them more applicable to research software.

The FAIR for Research Software Working Group (FAIR4RS WG) is well-positioned to be the community forum regarding FAIR principles for software, services and workflows [19]. The FAIR4RS WG was assembled by the Research Software Alliance (ReSA) [31], the Future Of Research Communications and E-Scholarship (FORCE11) [4], and the Research Data Alliance (RDA) [37]. They aim to represent many different stakeholders, including research software users, developers, maintainers, policy creators, managers of relevant infrastructure (e.g. publishers, archives), and research (software) funders [71]. In addition, mature wide-scale community engagement is promoted by basing it on the Center for Scientific Collaboration and Community Engagement (CSCCE) participation model which enables structured member engagement [107]. This working group created four subgroups that were assigned different tasks in order to support the creation of community-endorsed FAIR4RS principles [45, 46]:

1. Examination of FAIR Guiding Principles for research software from scratch [73]. A group of people, many of whom are involved in the FAIR4RS WG, previously published the position paper "Towards FAIR principles for research software" [79] aimed at developing a set of FAIR principles for research software. This subgroup took a fresh look at the same problem, and therefore planned to exclude authors of that paper[1]. They also did not look at previous research except for the original FAIR principles.
2. Application of FAIR Guiding Principles in other contexts like workflows and training material [78].
3. Definition of research software to provide context [60]. This definition is used throughout the study.
4. Review of new research related to FAIR Software since the release of the paper "Towards FAIR principles for research software", as well as the paper itself [47, 79, 103].

[1] In the end, authors of that paper were included. The original idea, however, was to exclude them.

Three additional subgroups were launched afterwards in September 2021 [44] to review adoption guidelines for FAIR4RS principles [81], (early) adoption support [83], and governance [65]. There are also related initiatives whose main focus does not lie on FAIR research software. However, since FAIR research software is related to FAIR data, they also contributed to this subject:

- The CURE-FAIR Working Group stated in their charter [102] that their objective is to "establish standards-based guidelines for curating for reproducible and FAIR data and code".
- FAIRsFAIR - Fostering Fair Data Practices in Europe - aims to supply practical solutions for the use of the FAIR data principles throughout the research data life cycle. Emphasis is on fostering FAIR data culture and the uptake of good practices in making data FAIR. The FAIRsFAIR Work Package 2 published an assessment report on "FAIRness of software" [59]. They focused on software as a research outcome.
- GO BUILD Pillar of GO FAIR US - they plan to develop a FAIR assessment framework for research software platforms, including badging which is a common practice in open source development [13].
- The Australian Research Data Commons (ARDC) created a FAIR self-assessment tool for data [1] and is in the development of a similar tool for software [2].

As previously mentioned, data can have privacy concerns which is typically not the case for research software. On the contrary, openness should be the norm for research software for many reasons related to facilitating open science, transparency, innovation through collaboration, reliability, and more. It should only be closed when appropriate reasoning for exceptional cases is given [62]. Chue Hong et al. mentioned that research software should be "as open as possible, as closed as necessary" [45]. It is also important for open science improvement to recognize research software as a first-class research output, similar to research data [28, 43, 56, 58].

Lamprecht et al. [79] published the first position paper that suggests modified, extended and additional FAIR principles for the application of these for research software. They explained why the application of FAIR principles for research software needs to take different aspects into account, why quality considerations go beyond FAIR and offer an analysis of where principles need adaptation or additional principles.

Software shares some characteristics like the possibility of having a Digital Object Identifier (DOI) assigned, which is one kind of a PID, or having a license added, but also has significant differences. Data are facts, while

software is the result of a creative process that is executable and usually consists of many dependencies with frequent updates, which leads to shorter lifetimes than for data. Research software can be made available on many platforms like digital repositories, archives, language-specific archive networks and more. In addition, research software is often FOSS which overlaps with FAIR regarding accessibility and reusability. FAIR principles, however, do not require openness. Certain data is sensitive and therefore require access control which is different from research software as this is part of the methodology which is expected to be shared.

They argue that there needs to be a distinction between the form and function of research software since quality considerations differ. FAIR principles cover the form with interoperability and reusability, but it needs to be mentioned that reusability also needs to take sustainability and the non-static nature of software into account, which is different to data. Maintainability improves the sustainable growth of software and includes modularity, understandability, testability, following guidelines and coding standards and can often be measured or quantified. Quality regarding function is more difficult to measure for many reasons, and there are ongoing discussions on whether applicable principles can be developed.

The original FAIR principles and the proposed FAIR principles for research software by Lamprecht et al. and Chue Hong et al. [46] can be seen in Appendix A. The complete comparison of the different principles can be found in the appendix of Chue Hong et al. [46].

Regarding **findability** for research software, the main concern is to ensure unambiguous identification by common search strategies. This includes search engines but also registries and repositories. In the following text, the principles proposed by Lamprecht et al. will be elaborated upon where necessary. Some principles are mostly rephrased to be suitable research software. For F1, a key difference is that research software needs a PID for all software versions to guarantee reproducibility. While Git offers code version control, GitHub, which is often used for publishing and active development, does not serve the purpose of persistent storage. Zenodo is often used together with Github to solve this. For F2, typical metadata that describes software should at least include

where to find specific versions, how to cite, authors of the research software, inputs and outputs, and dependencies. There are also ongoing projects that offer solutions for structured metadata annotation for software like codemeta [26]. It should also be noted that vocabulary provided by community-approved ontologies should be used where possible. An ontology is "a classification of categories describing a software artefact with explicit specifications of its entities and relationships in a certain domain of use" [61]. Having such an ontology helps to gain a shared understanding of used terms. For F4, it is important to note that there are three classes of registries and repositories:

- General ones like Zenodo, GitHub, and software archives like the Software Heritage project [16]
- Language-specific ones like CRAN or PyPI
- Domain-specific ones like the bioinformatics-specific BioConductor [8]

Research software should be published in a suitable registry or repository. This may be influenced by programming language or operating systems that are standard within the respective community.

Regarding **accessibility**, the original principles apply, while there is also the additional aspect of the ability to actually use the software. Alternatively, in other words, to access the functionality, which as a minimum, requires the availability of a working version. This also means that the software should be downloadable and runnable, as it might depend on data samples, (paid) registration, other (proprietary) software and more. The authors also mentioned that accessibility, interoperability and (re)usability are intrinsically connected for research software and consider aspects of installation instructions, dependencies and licensing as part of other principles. Changes are relatively little compared to the original accessibility principles. Mainly, they are rephrased to include a reference to software.

Regarding **interoperability**, a definition from the IEEE Standard Glossary of Software Engineering Terminology was provided: "ability of two or more systems or components to exchange information and to use the information that has been exchanged" [24]. This was complemented by a definition of semantic interoperability,

ensuring "that these exchanges make sense - that the requester and the provider have a common understanding of the 'meanings' of the requested services and data" [64]. Interoperability for research software is the most challenging high-level principle because software usually has high complexity. It is already the most challenging for data, which is usually less complex due to the static nature, while research software interacts with many components that also include other software and data. There were more modifications for the interoperability principles compared to the accessibility principles due to research software being live digital objects, unlike data. The authors provided three angles from which interoperability for software can be viewed at:

- For a set of independent but interoperable objects to produce a runnable version
- For a stack of digital objects that should work together to execute a given task
- For workflows which interconnect different standalone software tools to transform data

One requirement for interoperability is software metadata, which provides valuable information. This should include software versions, dependencies with versions, data types, input and output format, communication interfaces, and deployment options. Software portability, which refers to the ability to run software in different operating systems, is also part of interoperability. Software containerization tools like Docker or Rocket help in improving software portability. For I1, a key difference is that software source code is formal since it is written in a programming language. The focus of this principle lies in the ability to share input and output with other software, thus incorporating the terms machine readability and data exchange. For I2, this principle is split into two sub-principles. The authors consider software and data it operates on as separate cases. FAIR vocabulary should be used for both cases since FAIR software should operate on FAIR data. FAIRsharing.org provides a registry of vocabularies, data types, formats and schemas for software. For I3, the aim is the interconnection of data sets by semantically meaningful relationships, which is not directly applicable to research software. While software dependencies are somewhat similar, there is not much semantically meaningfulness required as the primary relationship is "dependsOn".

Regarding **reusability**, the main goal is to have reproducible re-use of software within different scenarios [36]:

- reproducing the same outputs as reported
- (re)using the software with different data
- (re)using the software for (unsupported) cases
- extending the software for additional functionality

Other relevant aspects are the previously explained software maintainability (documentation), the legality of reuse (licenses), and credit attribution to research software (citation). For R1.1, metadata should have separate licenses from software as they can be considered independent. License propagation is not needed for metadata. Software dependencies also make proper license management more complicated than for data since licenses of dependencies need to be considered. Machine readability becomes important to automatically determine incompatibilities of dependencies due to licensing. For R1.2, provenance refers to origin, source, and history of software and its associated metadata. Commonly included provenance data are contact information of the resource provider, publication date and location. Software versions are a minimum requirement to track the provenance of the software itself. Another aspect is the production of the software, for example, dependencies (see also I4S) and compilation of the software. Citation and contribution information is related to provenance.

Outside of the FAIR principles, other considerations are community-specific metadata schemes for software that will play an important role, and the need for a governance model for the FAIR principles to enable an open process for updating them. This should also be considered for the different domains within the scope of community discussions. The authors' work also provides the foundation for the development of metrics and maturity models to inform about software's FAIRness. The Netherlands eScience Center and DANS released five FAIR software recommendations [10, 82]. The Python package *howfairis* can also check compliance with the five recommendations for GitHub repositories [97]. Both of these resources will be updated to be more in line with the FAIR4RS principles [83]. These FAIR variables are also used in the SWORDS@UU framework and, there-

fore, in this study. However, the necessary level of FAIRness also depends on the type of software and should be discussed within the community.

The FAIR4RS-subgroup4 reviewed the position paper, including a survey regarding FAIR principles for research software and suitability of the FAIR principles proposed by Lamprecht et al. [47]. Having such a review allows further work to incorporate community feedback. One aspect that was often mentioned, especially in the question of what accessible and interoperable means in the context of research software, was sustainability. The principles had around 60 to 90 percent acceptance. Oftentimes, it was criticized what exactly is meant by the concrete principle or how it applies to research software, indicating that ambiguity needs to be minimized. There were further publications since the publication of the position paper that incorporated previous findings and community feedback to further develop FAIR principles [45, 46, 73].

The output of the previously mentioned FAIR4RS-subgroups was discussed by Katz et al. [72], which included a community consultation. This resulted in five concerns regarding the scope and interpretation of terms that need to be addressed:

- **General vs. specific principles**: There needs to be a balance between general and specific instructions. While actionable instructions are easier to act upon, they are also less long-lasting than general statements.
- **Long-term access to software**: Should this be relevant for FAIR Research Software Principles?
- **Defining research software**: There are two definitions that were considered, inclusive and exclusive. Inclusive refers to every software related to the research while exclusive refers to software only from reviewed publications.
- **Defining software**: Software can have many forms: scripts, executables, binaries, workflows and more. They suggest to define software as "A set of instructions that performs some action, either as source code (machine- and human-readable) or executable."
- **FAIRness of related research objects**: How should FAIRness of related objects like software dependencies and documentation be handled? Is FAIR recursively applicable?

Additionally, they looked at the different recommendations [10, 73, 79] regarding their differences and agreement on the FAIR principles. A summary of the findings can be seen in Table 2.1.

| Principle | Agreement | Discord |
|---|---|---|
| Findable | All agree about the foundational principle and mostly agree about the guiding principles. | Minor. Some aspects like software granularity are only considered in some recommendations and levels of detail differ. |
| Accessible | All agree about the foundational principle and that metadata should always be accessible even after software becomes unavailable. | Differences in interpretation and mentions regarding (metadata) accessibility, software, retrievability, authentication and authorization. |
| Interoperable | Some agreement in partial aspects in the definition of interoperability, but strong discord. | The three angles described by Lamprecht et al. are not agreed upon by the different recommendations. There is also disagreement whether controlled vocabularies like the Citation File Format are relevant. |
| Reusable | All agree that a clear license is essential, detailed provenance is helpful, software should be described with accurate and relevant attributes, and that potential use and reuse should be maximized while following software engineering best practices. | Only some recommendations mention maintainability and dependability. Some interpret reuse as a much wider term. There is also disagreement about the relevance of executability of software, encapsulation and interpretation of community standards. |

**Table 2.1:** Agreement and discord of different recommendations

Overall, interoperability and reusability are discussed and interpreted to varying levels of detail across the recommendations as captured in this community consultation. This largely stems from the different interpretations and meanings of these words that vary across research areas. There are differences even where the recommendations generally agree, like for findability. These findings indicate the need for researching characteristics for different domains to improve RSE practice.

The FAIR4RS Steering Committee drafted adoption guidelines [45] that were reviewed by the community [46]. For the development of the FAIR4RS principles, the original FAIR principles intention was taken as the starting point: "to maximize the added-value gained by contemporary, formal scholarly digital publishing" and

"to ensure transparency, reproducibility, and reusability" [104]. They also mentioned that the original FAIR principles might need to be reinterpreted, that FAIRness is not binary, that software can appear in different forms like executables and binaries, and that the FAIR4RS principles should be applied to all research software that is related to publications. Also, best practices in software engineering are relevant to the FAIR4RS principles since some practices directly improve FAIRness. The application of FAIR4RS principles is also the responsibility of software owners. The draft publication [45] was reviewed by the community and has been superseded by an updated version that incorporates community feedback on the proposed FAIR4RS principles for the release of version 1.0 [46]. These can be seen in Appendix A. The authors provided an accompanying text for each foundational and guiding principle that explains how to interpret the meaning. In the context of software, interoperability and reusability, as defined in the original FAIR principles, overlap. Therefore, interoperability focuses on data exchange between independent software. In contrast, reusability focuses on the ability of execution, inspection and understanding of humans and machines. The draft publication includes two sections that talk about implementation challenges and what is necessary to adopt FAIR4RS principles widely. The mentioned challenges hinder the adoption of the FAIR4RS principles. They are about metadata and identifier authority, metadata vocabularies and properties, software identifiers, identification targets, software structure complexity, FAIRness of related research objects, definition of accessibility and reusability, and the relation of openness and FAIR. They also provide a summary of what the European Commission deems necessary to support FAIR digital objects: metrics, incentives, skills and FAIR services [51].

Hasselbring et al. [62] reviewed the current state of FAIRness and openness of research software. The authors consider the pragmatic and infrastructure views of the variations of open science. The pragmatic view has the central assumption that "Knowledge-creation could be more efficient if scientists worked together". In contrast, the infrastructure view assumes that "Efficient research depends on the available tools and applications" [54]. These views require research software engineering to be sustainable, which requires RSEs. This is a relatively

new academic position. The RSE community displayed interest in promoting open science, which puts them in a perfect position to help researchers adopt FAIR and open software practices. A key obstacle in this endeavour is ensuring appropriate credit and recognition for the RSEs, which the authors recommend addressing via software citation and software observatories. Sustaining the communities, which include researchers, developers, maintainers, managers and active software users, is also essential. Research software has not reached a similar level of FAIRness as research data. Software versions are often not archived, for example, via Zenodo, but only available on GitHub, which is not made for archiving. Journals often allow to add supplementary material to publications, but it is often not further reviewed or explained and therefore does not support reuse. However, software journals exist where research software might be published, for example, the Journal of Open Source Software or the Journal of Open Research Software [14, 15]. There is also the possibility of sharing research software services to allow for direct reuse of the software without the need for installation procedures [7, 20]. They also state that research software has often been of subpar quality. This includes (no) testing, lacking documentation and coding standards which is a result of scientists not being judged based on the quality of the software that enables their publications. However, there have been initiatives including but not limited to several ACM conferences to start an artefact evaluation process, as well as the "Most Reproducible Paper Award" by ACM SIGMOD. The authors provide recommendations to address the issues of research software not being properly published for scientific reward and proper citation along the FAIR principles. Challenges regarding **findability** are methods of software citation and software retrieval. A key challenge is appropriate software metadata. Some solutions regarding software citation and identification were previously mentioned [26, 52]. Software artefacts should be published with preservation in mind to increase **accessibility**. GitHub alone does not support preservation. A combination of GitHub for use, reuse and active involvement, and Zenodo for archival and reproducibility, achieves this [12]. Another example would be the Software Heritage archive [50]. Established software and data standards should be followed to increase **interoperability** and may be part of artefact evaluation processes. In

addition, virtualization through Docker, for example, and online services improve interoperability and reusability across platforms. Artefact evaluation processes should review replicability, reproducibility, and reusability of research software to increase **reusability**. Modular software allows for reuse of parts of the software, and good engineering practices ensure that other software can be built on top by others. These include, for example, proper documentation, providing testing frameworks, and test data for continuous integration. Based on their experience with their analysis of the relationship between research publications and research software, the authors proposed the deployment of decentralized research software observatories, which would support research software retrieval and analysis. Such an observatory would provide support to open science research and encourage best practices among research communities. Additionally, it would allow exploring opportunities and challenges of cataloguing research software. Two critical components of such an observatory are support for metadata that allows classification of research software and citation to enable FAIRness. In that regard, they also mention the Research Software Directory [98] as a related system. However, this system's focus is on repeatability and less on FAIR and open principles. A first observatory for life sciences is already being developed [84, 88].

Anzt et al. [34] described the status and challenges of research software sustainability and suggested possible improvements with a focus on the German landscape. Common challenges for creating sustainable research software were described. They identified different stakeholders and where their interest stems from, ranging from the general public, geopolitical units, industry and independent developers to RSEs, domain researchers and research funding organizations. Since this study's goal is to improve an organization, it is of primary interest what motivates RSEs, research leaders, domain researchers, research funding organizations, and libraries which can be part of an organization. The common arguments for the motivation of sustainable research software for these people of interest are intrinsic interest in developing more sustainable and higher quality software, visibility and reputation, spending more time and funding on actual research and less on (re-)creating software, compliance with best practices, and reuse in other areas. The authors proposed a set of evaluation criteria for funding sus-

tainable research software based on previously proposed evaluation schemes. The criteria are either hard criteria that everyone should fulfill or soft criteria that depend more on specific community standards. These are about usage and impact, software transparency and quality, and software maturity and are also part of a usual Software Management Plan (SMP) [21]. It makes sense to focus on sustaining research software that is expected to have high usage and impact, good quality and high maturity to utilize available RSE resources best.

Gruenpeter et al. [61] reviewed and analysed available FAIR principles for research software and presented recommendations for constructing these principles. They stated that it needs to be taken into account that researchers already face significant challenges regarding research software development and maintenance. They also mentioned the need for training to produce FAIR and citable software and that issues arise due to different interpretations for data and software, for example, regarding reuse and interoperability. The CURE-FAIR Working Group [87] also reviewed computational reproducibility challenges and has determined sociocultural factors as one of the main challenges. This includes insufficient training and skills. Martinez-Ortiz et al. [83] identified early adopters of the FAIR4RS principles where common actions taken include training, metrics, guidelines and infrastructure [81]. Especially the need for metrics to better quantify FAIRness has been emphasized, and a working group for metrics development was created [3, 9, 35]. Pico et al. [88] proposed first indicators for quantitative measurement of FAIRness.

During consultations and presentations in the context of the SWORDS@UU framework with different stakeholders, we noted that researchers indicated their approval of FAIR principles for research software. However, their issue often was that they did not know how to make their research software more FAIR. The main reason for this is that the principles are too abstract and not directly actionable, which further illustrates the need for appropriate training and resources. Researchers are typically not formally educated software engineers, and as mentioned, best practices often overlap with FAIR principles. As an experiment, we contributed to research software repositories to improve their FAIRness, which was well received.

Since the first publications of FAIR principles, various projects and initiatives have supported the facilitation of FAIR research software [2, 26, 49, 81, 93, 97, 100]. What can be seen from the existing literature regarding FAIR research software is that some things are generally agreed upon, which includes the provision of a license, rich metadata, the need for quantifiable metrics, possibility for citation, infrastructure, and training. On the other hand, there are differing opinions regarding interpretations of suggested principles, scope, and definitions. This indicates that there exists ambiguity that needs to be cleared or that terms are understood in different contexts depending on the domain.

## 2.5 GitHub research software analyses regarding FAIRness and research domains

There have been many analyses on GitHub data [48]. However, none of them focused on developing recommendations of FAIRness for RSE practices on an organizational level. Their focus often lies on highly active projects, which is not our focus [38, 40, 66, 76, 80, 91, 95]. Research software might not have these characteristics. Since we are interested in the differences between research domains, we looked at GitHub analyses that concern themselves with domains. Additionally, analyses that suggested metrics regarding openness and sustainability were inspected, as these topics are related to FAIRness. Due to the topic of FAIR research software being relatively new, there are not many publications regarding FAIR research software analysis on GitHub compared to the amount of general GitHub analysis.

Gharehyazie et al. [57] found that code reuse across projects in GitHub for FOSS code is significant and more common within domain boundaries. These findings all point towards existing cross-domain differences and show that project characteristics differ between different areas.

Russell et al. [94] mentioned that the background of researchers is very diverse, leading to a unique culture

around programming within Bioinformatics, where repositories tend to be small with few contributors. They capture highly varying repositories for their analysis. Key findings include that around half of the repositories did not have a license and a lack of reuse.

Hasselbring et al. [63] conducted an analysis of research software in the research areas of computational science and computer science to provide recommendations to increase FAIRness and openness. They noted that the emphasis in computational science lies in reproducibility, while the emphasis in computer science lies in reuse. They follow that publication behaviour varies between research areas. The average life span of research software is defined as the time from the first to the last commit activity. This metric greatly differs between computer science and computational science with a 5-year median and 15-day median, respectively.

Pico et al. [88] proposed a software quality assessment framework for FAIRness using weighted indicators as a scoring tool that combined different (meta)data sources, including GitHub. This was tested on software within the life sciences, thus depicting an initial landscape of that domain and providing the first steps toward an observatory. As we have seen from Katz et al. [72] in the previous section, there is unclarity regarding general and specific principles. This study is helpful as it provides specific and concrete things that researchers can scrutinize to improve FAIRness of their software. Their analysis found that research software in life sciences is highly findable, moderately accessible and reusable, and barely interoperable with their defined weights of the indicators. Findability being the highest makes sense, as otherwise, the research software would not be able to be part of the dataset. Interoperability also proved difficult to be translated into measurable indicators, raising the question of whether the measurability is the problem or if low interoperability is an actual finding. This is in line with what Lamprecht et al. [79] mentioned, that interoperability is the most challenging high-level principle. Many indicators are related to specific resources within the life sciences, which illustrates that domain-specific indicators might be necessary. This becomes apparent if one would like to assess if accepted ontologies were used due to domain specificity or if the software is found in a domain-specific registry.

# 3 Research method

We conducted an exploratory data analysis to gather more insights into the research software landscape [99]. The study is based on a GitHub dataset and is quantitative. We analyzed the data with the help of Python [101]. Details of the data collection are described in Section 3.1. Section 3.2 explains the data analysis, which included the data exploration and statistical methods. Section 3.3 illustrates the validity of this study.

## 3.1 Data collection and labelling

The data were collected through the SWORDS@UU framework [49], which collects users, their repositories and variables in three corresponding phases. Any additional code that is not part of the SWORDS@UU framework, instructions for reproducing the research, and the accompanying data were published under https://github.com/kequach/Thesis-Mapping-RS. The collected FAIR variables were included in the SWORDS@UU framework in a new release. Supplementary steps for this study were marked with an asterisk in the corresponding figures for each phase.

### 3.1.1 User collection

In phase one, the framework collected identified UU employees through different search strategies: GitHub search API [5], UU employee pages API[1] [23], PURE [18] output from UU, and the PapersWithCode.com API [17]. The complete phase one that was adjusted for this study can be seen in Figure 3.1.

After the potential GitHub usernames were collected, they were merged and enriched with the actual GitHub API data. The filtering and labelling of users are semi-manual tasks. The filtering excludes all non-employees without an employee profile page. In other words, anyone that can not be retrieved via the API of Utrecht University

[1] The API is undocumented. See [90] for examples on how to interact with the API. [23] shows the web interface of the API.

was excluded, except for organizational accounts like ASReview. Students were included in case they could be retrieved via the API. This also implies that former employees and University Medical Center Utrecht (UMCU) employees were not included in this study. Organizational accounts that depict larger collaboration groups that work on a national level like CLARIAH are included for comparison but are not considered part of Utrecht University for the analysis. The labelling with additional information about faculty membership was done in addition to the SWORDS@UU framework. The faculty and collaboration group information was added to the repositories in the next phase. The filtered users are described as *irrelevant* users hereafter.

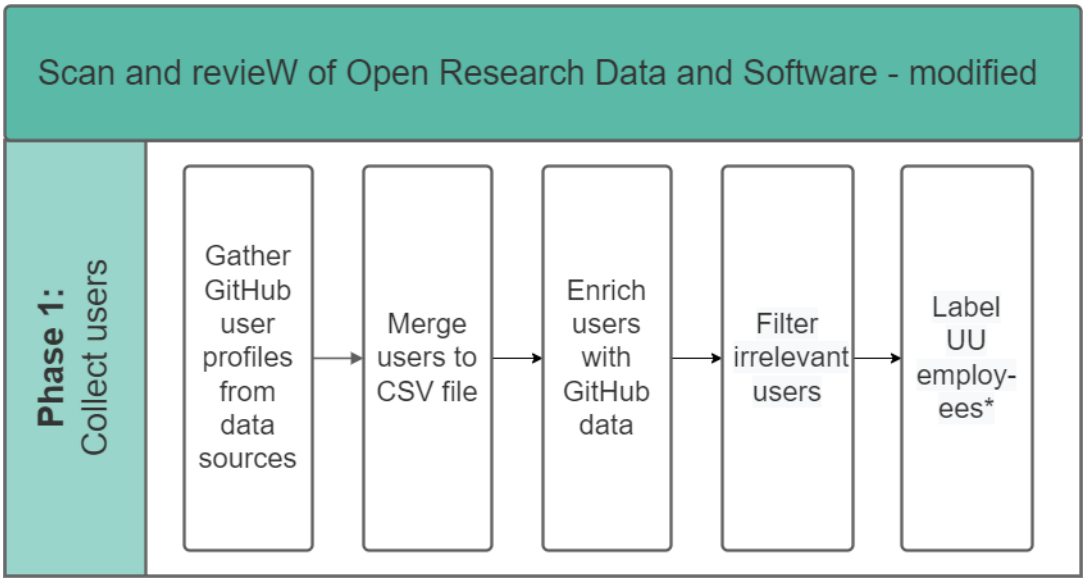


**Figure 3.1:** The modified first phase of the SWORDS@UU framework. Users are collected, filtered and labelled [49].

The collected users of the steps and search strategies can be seen in Figure 3.2. All users that were retrieved from the different search strategies during the first step were merged, which left us with 628 unique users. From these, we filtered out 428 irrelevant users, which resulted in 176 identified GitHub users employed at UU.

The 176 UU-related users are further explained in Figure 3.3. Automatic labelling based on provided GitHub metadata and information from the employee pages labelled 129 out of 176 users that needed to be manually verified. The remaining users and the organizations were manually labelled. This process is time sensitive as

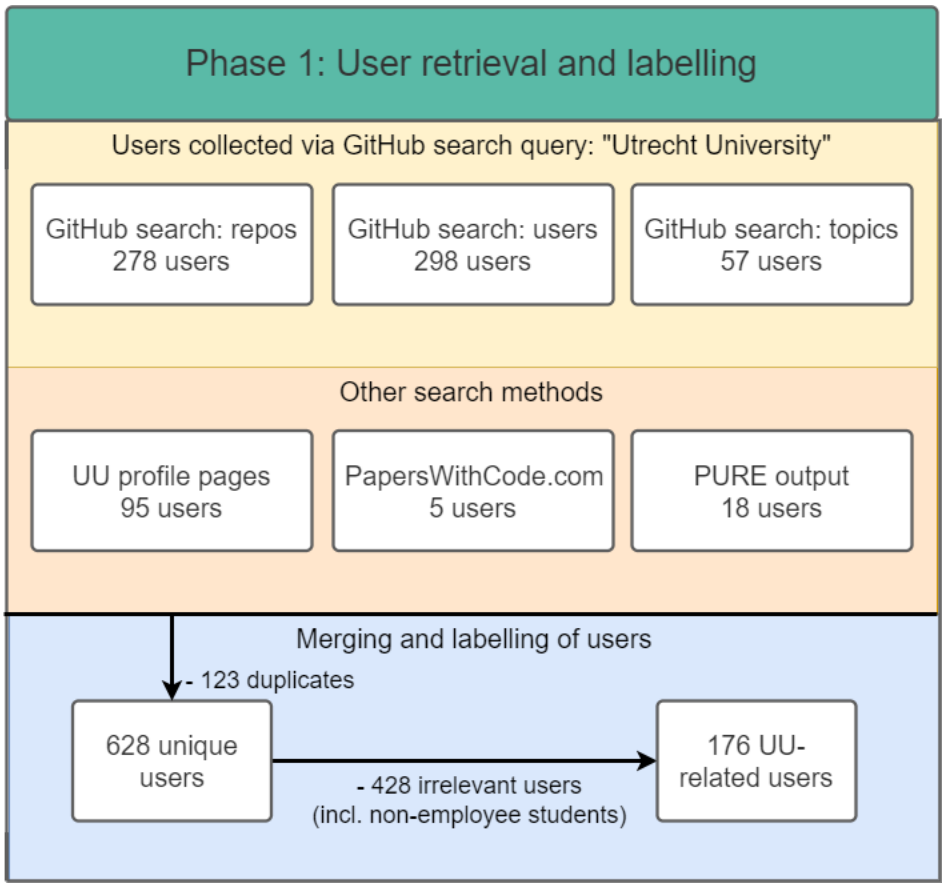


**Figure 3.2:** Results of the non-modified SWORDS pipeline for phase one.

employees leave and join the university at any given time. The users were collected and labelled during the first two weeks of July 2022. Therefore, this process is naturally not utterly reproducible due to the external API dependency.

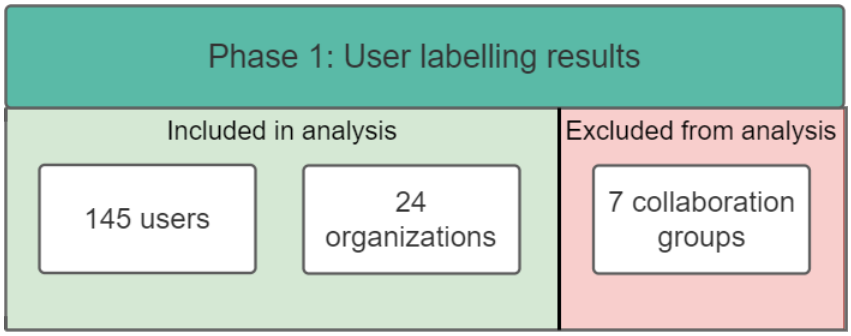


**Figure 3.3:** Results of the user labelling for phase one.

### 3.1.2 Repository collection

In phase two, the repositories were collected from the users that were the output of phase one. The complete phase two that was adjusted for this study can be seen in Figure 3.4. Filtering consisted of automated filtering of forks and GitHub.io repositories. Forks are copied repositories that are usually used to create pull requests for the original repository. GitHub.io repositories are usually used for documentation or personal websites and not software. Filtering non-research software projects is highly important since most repositories are personal [48, 68]. Manual labelling of remaining repositories followed afterwards. The repositories were collected on 18.07.2022 and subsequently labelled until 10.08.2022. This was done in alphabetical order based on username first and repository name second in two iterations, where unclear repositories were left to the second iteration. An overview of the used repository type labels can be found in Appendix B. This left us with 823 research-related repositories and 698 non-research-related repositories that are kept for comparison and classification.

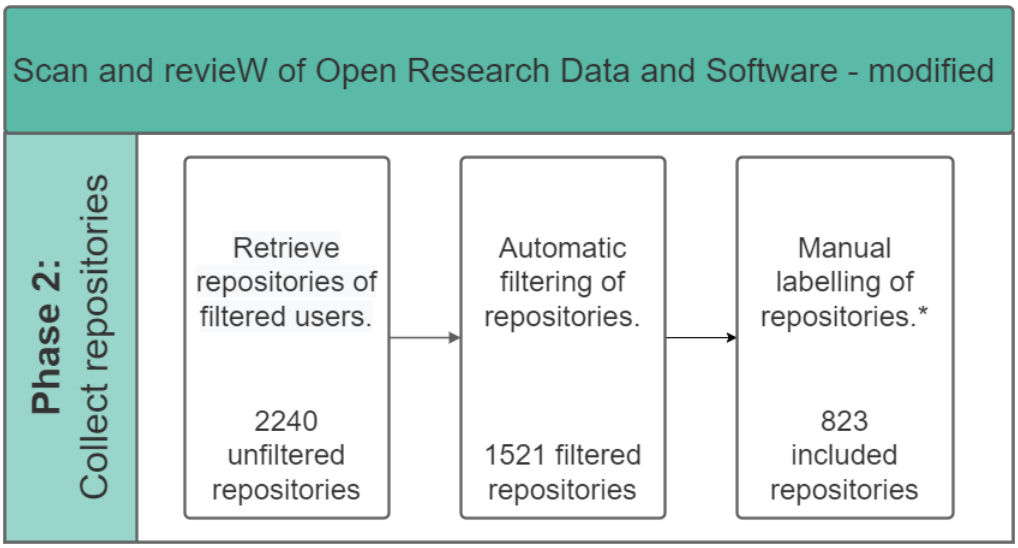


**Figure 3.4:** The modified second phase of the SWORDS@UU framework. Repositories from identified users are collected, filtered and labelled [49].

In addition to the repository type labelling, we also labelled each repository with the corresponding faculty, which is determined by the owner of the repository. There are seven faculties and two support departments:

- Faculty of Geosciences
- Faculty of Humanities
- Faculty of Law, Economics and Governance
- Faculty of Medicine
- Faculty of Science
- Faculty of Social and Behavioural Sciences
- Faculty of Veterinary Medicine
- University Corporate Offices
- Utrecht University Library

For labelling, manual verification is always needed. As this study only considered research related to UU, it was necessary to find out during what time period an employee was employed if the repositories might have been developed at another organization. As such, repositories that were developed before or after employment at UU were filtered out to the best of available knowledge, e.g. added CVs on employee pages or personal websites. Also, it was required to understand the researchers' subject to better differentiate between research-related and non-research-related projects.

TNO [22], an independent research organisation, was not treated as a different entity since researchers can be employed at both TNO and UU. Therefore, being an employee of TNO does not exclude a user from our dataset. Even after extensive investigation, some repositories were unclear whether they could be considered research-related or not. In these cases, the author was contacted to clarify, and all requests were answered. In total, this concerned only six users and ten repositories. For some repositories, it was not possible to clearly distinguish between the labels *non-rs, student work, and irrelevant*, as they are not mutually exclusive. However, that is not an issue since further analysis mostly excluded them, and classification only considers whether a repository is research software or not.

### 3.1.3 Variable collection

In phase three, the howfairis variables, nested GitHub variables, and additional FAIR variables were collected. The complete phase three that was adjusted for this study can be seen in Figure 3.5. The howfairis variables were created to check the compliance with the FAIR recommendations from the Netherlands eScience Center and DANS [10]. While the first four variables are related to FAIRness, the *has checklist* variable attempts to cover related topics, such as software engineering best practices. The variable only checks for one very extensive checklist, which might not be appropriate for all kinds of research software [6]. Nested variables include information about contributors, used (programming) languages, repository topics, and the README's content. These were collected in separate datasets to achieve a tidy data structure. Additional FAIR variables that were computed based on available GitHub variables were identified with the help of Martinez et al. [81, 97] and related GitHub analyses [63, 88, 94] and can be seen in Table 3.1, together with the howfairis variables. For determining to which FAIR principle a variable relates to, we considered a combination of the principles by Lamprecht et al. [79] and Chue Hong et al. [46] that can be seen in Appendix A. The additional step *Gather and compute additional FAIR variables* was incorporated into the SWORDS@UU framework. The variables were gathered on 12.08.2022 and 13.08.2022.

The howfairis tool outputs five binary FAIR variables. Table 3.1 also describes the additional FAIR variables that were gathered based on existing literature. These were not part of the SWORDS@UU data collection yet and were incorporated into SWORDS@UU as part of this study. The FAIR variables from Russell et al. [94] and Hasselbring et al. [63] should serve as a proxy to measure the FAIRness of the repositories. They are about openness and sustainability, which are related to the FAIR principles. For the FAIR variables from Pico et al. [88], we consider incorporating the ones that are applicable to only GitHub data. Many indicators are always valid for GitHub data and, therefore, are not applicable. The implementation of metrics retrieval also differs

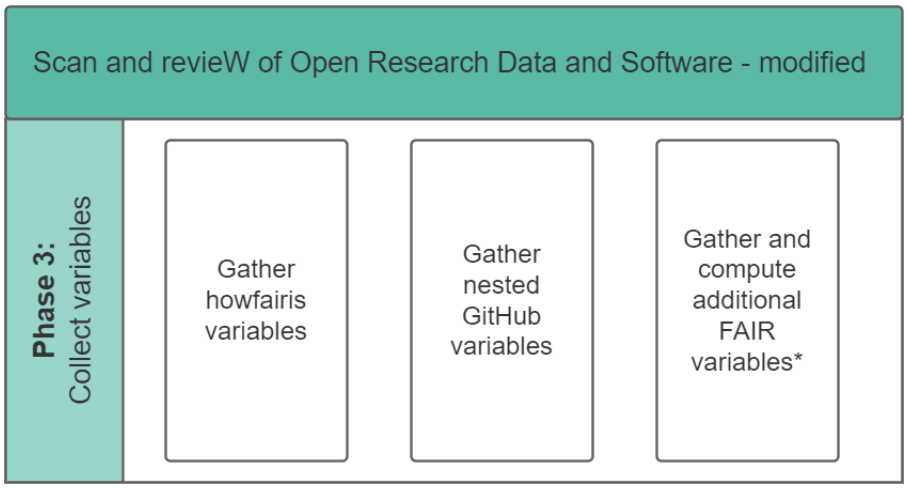


**Figure 3.5:** The modified third phase of the SWORDS@UU framework. Variables from identified repositories are collected and computed [49].

from the implementation of the references since the quality of these can be improved. As an example, the *version control usage* by Pico et al. only counts versioning with the form of X.X as valid. This does not include semantic versioning, which has the form of X.X.X [89]. In addition to the FAIR variables, we included the metrics in Table 3.2 for data analysis. Each metric represents the natural count of occurrences unless otherwise described. Life span is also considered a metric since it is a numeric variable. We also take a deeper look at the license, languages, and topics.

## 3.2 Data analysis

To validate subquestion 2, we looked at the Jaccard similarity coefficient [77] of the howfairis and new FAIR variables derived from literature and their percentages for research and non-research software. In our case, Jaccard similarity measures pairwise how many of the FAIR variables in two repositories are equal and how many are different. It is, therefore, an appropriate measurement for boolean variables. The assumption here is that there should be a high similarity to existing FAIR variables if they are suitable supplementary variables. This is the preferred measurement over the usual correlation since these variables are all binary. The life span was excluded

| Name | Description | FAIR |
|---|---|---|
| Repository open? | Is the repository open [97]? | F |
| Has license? | Is there a license [97]? | R |
| Is registered? | Is the repository registered in a registry [97]? | FI |
| Has citation? | Is there citation information [97]? | R |
| Has checklist? | Is there a checklist[2] [97]? | FAIR |
| Correct vcs usage? | Does the repository use version control correctly? This is measured by whether the repository's first and last commit are on the same day [94]. | RS |
| Life span | The length of time between the first and last commit activity [63, 94]. Not considered as FAIR variable but as an additional metric. See Table 3.2. | S |
| Repository active? | Was there a commit within the last 365 days [63]? | F |
| Has install instructions? | Are there installation instructions available [88]? Checked by mentions of **install** or **Docker** in the README file. | R |
| Has example usage? | Are there examples available [88]? Checked by mentions of **usage, getting started, quick start, example, tutorial** in the README file. | R |
| Has contribution guidelines? | Are there contribution guidelines available [88]? Checked by mention of **contribut** in the README file. | A |
| Has tests? | Is there a tests folder available [88]? | R |
| Version identifiable? | Is there a scheme to uniquely and properly identify the software version [88]? Valid schemes have the form of **X.X** or **X.X.X**. | F |

**Table 3.1:** Identified variables with descriptions. The FAIR column explains to which principles the variable is related to. S stands for sustainability. Except for life span, all variables are binary.

[2] The used version (0.14.1) only checks for the OpenSSF Best Practices checklist

| Name | Description |
|---|---|
| Stargazers | Represents the number of people that have starred a Github project. Starring a project can indicate that a user likes the project. It can also be used to bookmark a project, since starred projects are saved. The amount of stargazers can be used as a metric to measure popularity. |
| Issues | A way to keep track of the tasks, enhancements and bugs of the project. They can be discussed in a thread by users and developers. Each repository can enable their own issue page. An issue can be open, for example when a new bug is found, or closed, when it is solved. This shows the amount of open issues a repository has. |
| Forks | A fork is a copy of a repository for another user. |
| Size (in MB) | The size of a repository in MB. |
| Contributors | Contributors refers to the users that have contributed to a repository via commits and pull requests. The number of contributors gives information on how many people put effort into the repository. |
| Languages | Languages refers to the used languages in a repository. These can be programming languages, markup languages, shell scripts and more. |
| Topics | Topics refer to the topics a repository is associated with. These are self-assigned. |
| Life span (in days) | Life span refers to the time between the first and last commit in days. |

**Table 3.2:** Used metrics with descriptions.

for this part as it is a metric. We also computed descriptive statistics of the metrics to help answer subquestions 3 and 4: this included minimum, maximum, mean, median, skewness, and kurtosis. Additionally, we looked at the descriptive statistics of metrics, details of FAIR variables, and details of license, language, and topic usage for each faculty. A FAIR score is computed based on available FAIR variables. This is used as a proxy for quantifiable metrics, as most of the variables are binary and therefore not metrics.

For subquestion 3, we additionally conducted multiple univariate Kruskal-Wallis tests with applicable variables, which are all metrics, followed by a post hoc analysis with Dunn's tests. The Kruskal-Wallis test represents

the non-parametric alternative for one-way Analysis of variance (ANOVA). An ANOVA tests equality of means, while a Kruskal-Wallis test compares mean ranks. It is a more general test and less powerful. However, ANOVA assumes normal-distribution, which our variables do not fulfill. The Dunn's test compares the mean of each group pairwise and calculates which groups are significantly different. An alternative approach would be to use a non-parametric multivariate analysis of variance, of which multiple methods exist [33, 69]. There are different justifications for choosing either approach, and they address different research questions [67]. Some drawbacks are that multiple univariate tests ignore the increased precision of pooled variance estimates, decreasing inference reliability, and the estimation of the correlated error structure, which a multivariate model takes into account [30]. However, as it is of interest to us in which metrics the differences exist, a multiple univariate approach is more applicable. Huberty and Morris [67] also mention that multiple univariate analysis is applicable for exploratory research.

Subquestion 5 was answered with the help of two machine learning model classifications: *logistic lasso regression* and *random forest*. These were trained and tested on 80% and 20% of the data, respectively. The models included all metrics and FAIR variables. Metrics were scaled to values between zero and one. This allowed us to draw comparisons between variables from the logistic regression coefficients, as all variables are now on the same scale. We used the Python package scikit-learn [86] for this part of the analysis. For describing the models, we will use the term *features* for independent variables and *class* for the categorization of whether a repository is considered research software or not. We further excluded the feature *repository open*, as this should usually be true for all collected repositories. However, some repositories were either deleted or changed to private between the repository and variable collection, leading to some repositories having a non-true value. These were also excluded from the classification. We looked at feature importances to determine the most useful ones for each model. Feature importance for logistic regression is measured by coefficients. For random forest, it is measured by permutation feature importance. An alternative measure would be impurity-based feature importance. However, this method

is biased towards high cardinality features, which we try to avoid. Additionally, we measured the models with the performance measures accuracy, precision, recall, and F1-score. We compared the scores with a majority class prediction which we named *chance*.

A common topic for consideration while working with generalized linear models, which include logistic regression, is multicollinearity. This refers to the features in such a model having a high correlation. Multicollinearity leads to more unreliable estimates of the coefficients. A common way to address this is by inspecting whether the variance inflation factor (VIF) has a value below five [29]. Table 3.3 shows that none of the features have a VIF above five. The logistic lasso regression is capable of handling highly correlated features by shrinking coefficients of these towards zero, effectively serving as a feature selection. Logistic lasso regression, as implemented by scikit-learn [86], has one hyperparameter $C$ that was tuned via cross-validated randomized search with accuracy as the scoring function. This hyperparameter serves as an inverse of the regularization term, effectively determining the strength of the regularization. A low value equals high regularization, while a large value effectively equals no regularization. As such, we used a randomized search in a log uniform distribution on the training data, ranging from 0.00001 to 100 with 500 iterations, which can be seen in Figure 3.6. This lead us to choose a C-value of 2.545.

The random forest model had two hyperparameters that were considered for grid search tuning: *maximum number of features* and *number of trees*. While there are many more possible hyperparameters to tune, the default tends to do well for many cases. The grid search concluded with the maximum number of features being the number of all features and a maximum number of trees being 500, which was the maximum search value.

| Feature | VIF |
|---|---|
| Stargazers | 3.87 |
| Issues | 2.21 |
| Forks | 4.46 |
| Size (MB) | 1.23 |
| Contributors | 3.29 |
| Languages | 2.93 |
| Topics | 1.56 |
| Life span (days) | 2.35 |
| Has license? | 2.42 |
| Is registered? | 1.41 |
| Has citation? | 1.42 |
| Has checklist? | 1.16 |
| Correct vcs usage? | 3.65 |
| Repository active? | 1.88 |
| Has install instructions? | 2.08 |
| Has example usage? | 1.77 |
| Has contrib. guidelines? | 1.31 |
| Has tests? | 1.70 |
| Version identifiable? | 1.74 |

**Table 3.3:** Variance inflation factor values

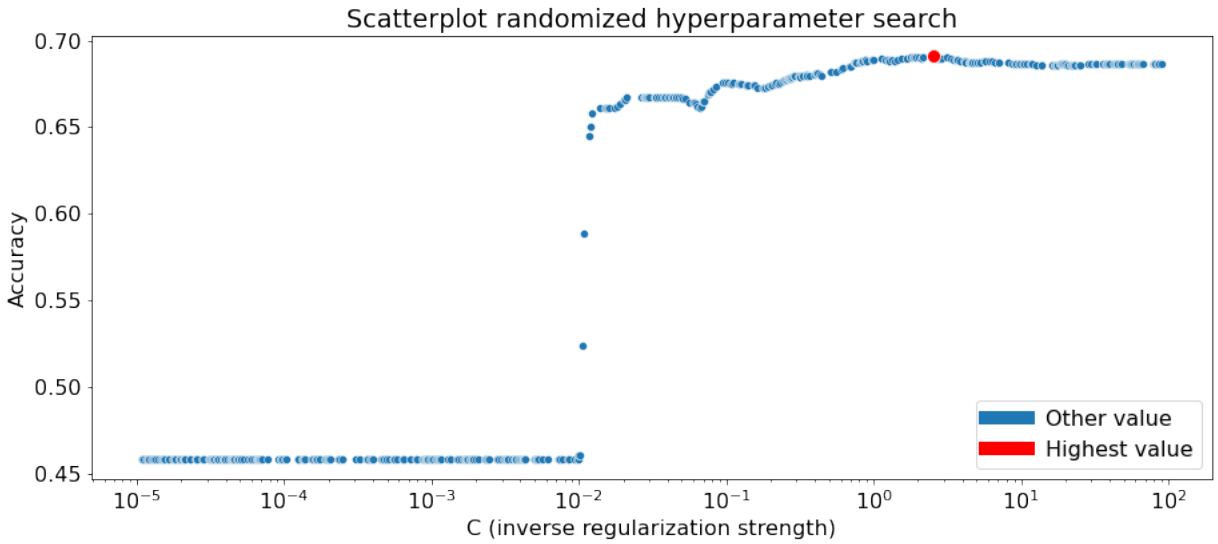


**Figure 3.6:** Results of the randomized search with a loguniform distribution for values from 0.00001 to 100 and accuracy as the performance evaluation metric.

## 3.3 Validity

There were several threats concerning the study. Contributions to research software that UU-affiliated users do not own are not captured. This means we could not capture if a researcher would contribute to existing open research software. The user search process may be flawed, resulting in a biased capture of UU employees. A sole focus on GitHub data may also lead to a similar bias in the data. However, existing literature indicated that this is negligible. The used howfairis tool [97] also has flaws. Registering a repository in a registry or having a checklist is not appropriate for all kinds of research software. Especially the checklist can often be too convoluted for simple research software, such as research scripts. Due to the amount of captured repositories, manual labelling of these might also be error-prone, especially considering the opinionated discussions regarding the definition of what qualifies as research software [60]. The daily supervisor Jonathan de Bruin reviewed the created labels to minimise possible mislabelling. However, a more significant threat to any GitHub analysis validity would be to include personal use repositories [68], as they are not the focus of this study.

These threats are an inherent part of the method used. Nonetheless, the results benefit academia, as stated in Chapter 1. The findings help to improve the RSE practice with concrete recommendations and novel findings. The method can also be adapted for other organizations as it is publicly available [90] and thus lays the groundwork for similar studies that may also improve the proposed method. Additionally, the created dataset will be openly shared for others to reuse.

# 4 Results

This Chapter illustrates the results of the data exploration and the statistical analysis. First, we describe the retrieved data on a user level in Section 4.1. Next, we look at the repository characteristics, including all metrics and details about license, language, and topic usage in Section 4.2. This is followed by inspecting the FAIR variables in more detail in Section 4.3. Lastly, we look at the statistical analysis of the dataset, which includes the statistical tests in Section 4.4 and the classification of research software against non-research software in Section 4.5.

## 4.1 Users characteristics

The number of collected users per faculty can be seen in Figure 4.1. *UtrechtUniversity* represents the GitHub repositories of the organizational user *UtrechtUniversity*. Figure 4.1 shows no users within the collected data from the *Faculty of Medicine*. It is expected to have no users related to the Faculty of Medicine as they are employed by the UMCU and, therefore, not part of this study. It is also expected that the *Faculty of Science* would have the highest number of users, as they are most likely to extensively use GitHub for research and non-research purposes.

The number of repositories per user per faculty can be seen in Figure 4.2 as a swarm plot with an overlapping box plot. What can be seen is that most users have a small number of repositories. In contrast, only a few have a vast number of repositories. There is also no dominant faculty for the users with many repositories.

Figure 4.2 shows the same plot with the user type instead of faculty. It shows that except for one user, it is usually an organization or collaboration group that has a large number of repositories. It also seems logical if this information is digested together with Figure 4.2. While there can be many users from a few faculties, there can naturally only be a few organizational-level users for each faculty.

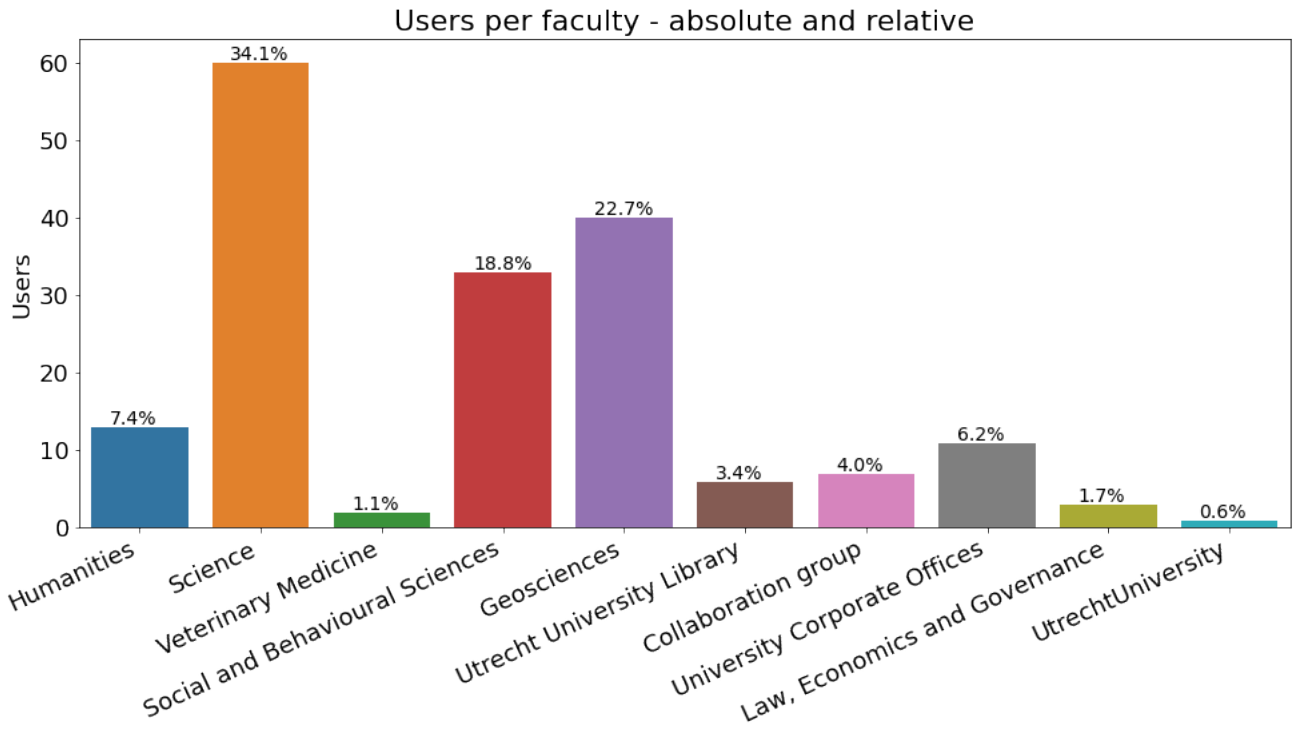


**Figure 4.1:** Number of users per faculty - both absolute and relative numbers. The y-axis shows the count, the number on top of a bar shows the percentage.

Table 4.1 shows the statistical measures of metrics for the collected users. The 25th quantile and median show that most users have a small number of followers and users they are following. The high values for skewness and kurtosis also indicate this. The number of repositories is similarly skewed but to a lesser extent, which the median also points towards. Additional box plots for number of followers and following can be seen in Appendix C.

| | Public repositories | Followers | Following |
|---|---|---|---|
| Minimum | 0.00 | 0.00 | 0.00 |
| 25th percentile | 2.00 | 0.00 | 0.00 |
| Mean | 13.02 | 10.32 | 5.44 |
| Median | 6.00 | 2.00 | 1.00 |
| 75th percentile | 14.00 | 9.00 | 4.00 |
| Maximum | 126.00 | 210.00 | 103.00 |
| Skewness | 3.31 | 5.16 | 4.32 |
| Kurtosis | 13.19 | 30.98 | 24.24 |

**Table 4.1:** Descriptive statistics of numeric variables for all users

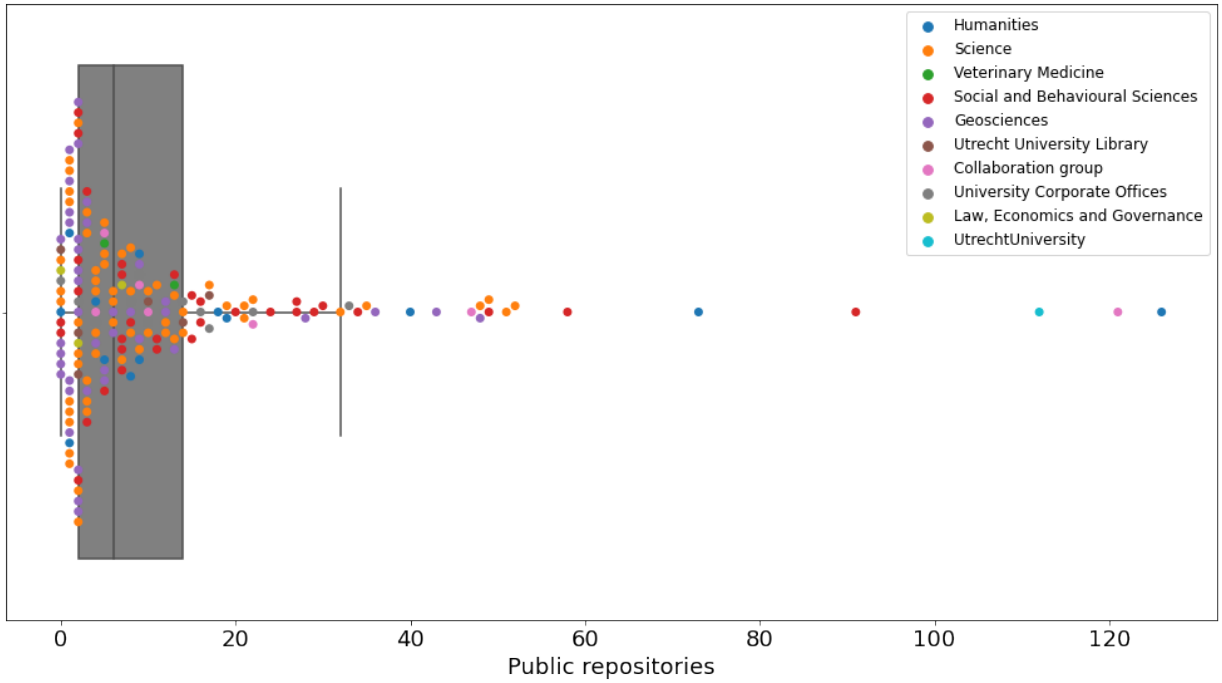


**Figure 4.2:** Number of public repositories per faculty.

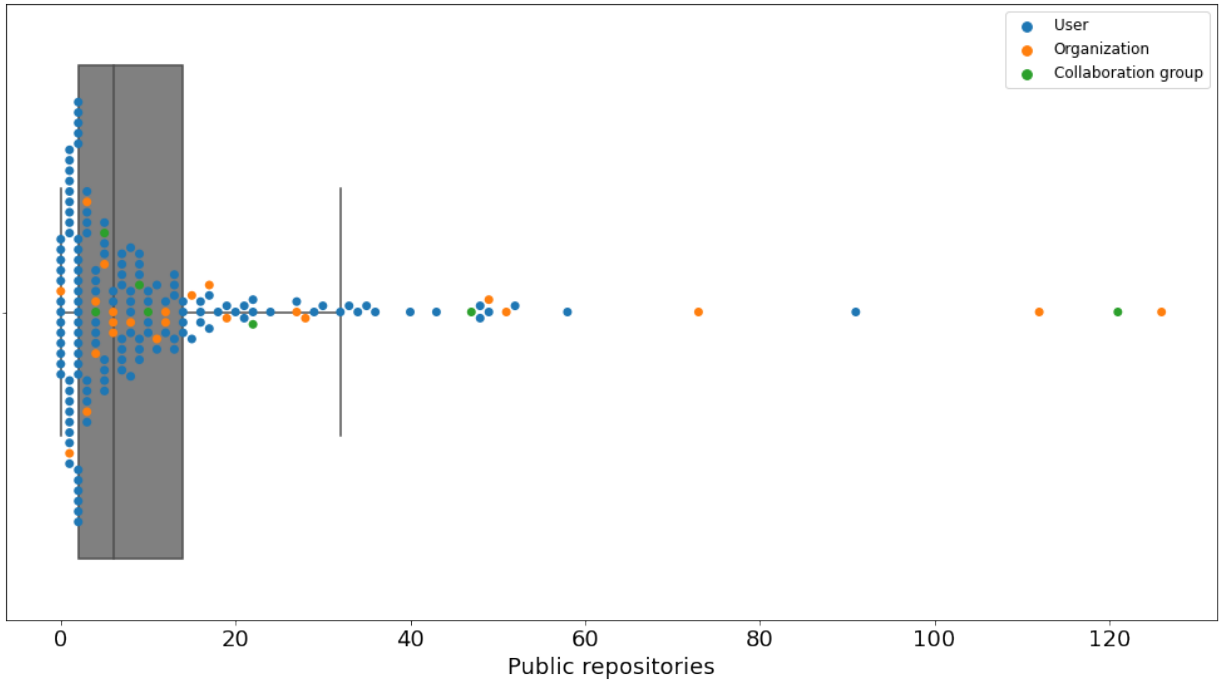


**Figure 4.3:** Number of public repositories per user type.

## 4.2 Repository characteristics

The number of collected research software repositories per faculty and repository type can be seen in Figure 4.4. Interestingly, the *Faculty of Law, Economics and Governance* has no open research-related repositories. We also see a low number of research-related repositories for the *University Corporate Offices*, *Utrecht University Library*, and the *Faculty of Veterinary Medicine*.

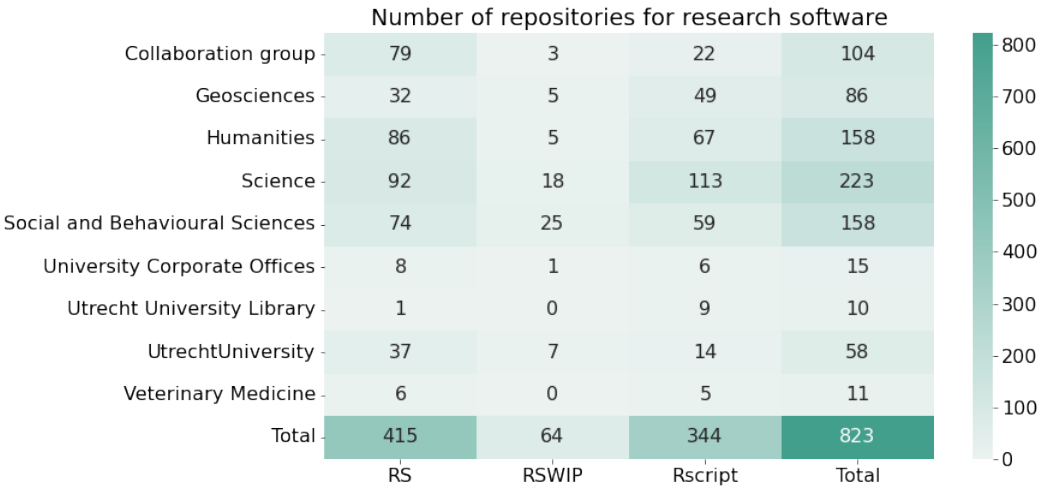


**Figure 4.4:** Number of research software repositories grouped by faculty and repository type.

Regarding non-research software, Figure 4.5 shows that most non-research software is related to documentation, general non-research software, and workshops. This is logical, as most repositories within *docs* and *workshop* still relate to academia but are more relevant for archiving or educational purposes. Non-research software most often depicted either private projects or projects to get familiar with new programming languages, concepts, or algorithms.

*University Corporate Offices* and *Utrecht University Library* are support departments and will be grouped together with the group of *UtrechtUniversity* for further analysis under the name of *Support departments*. The faculties *Veterinary Medicine* and *Law, Economics and Governance* are excluded from further analysis due to the

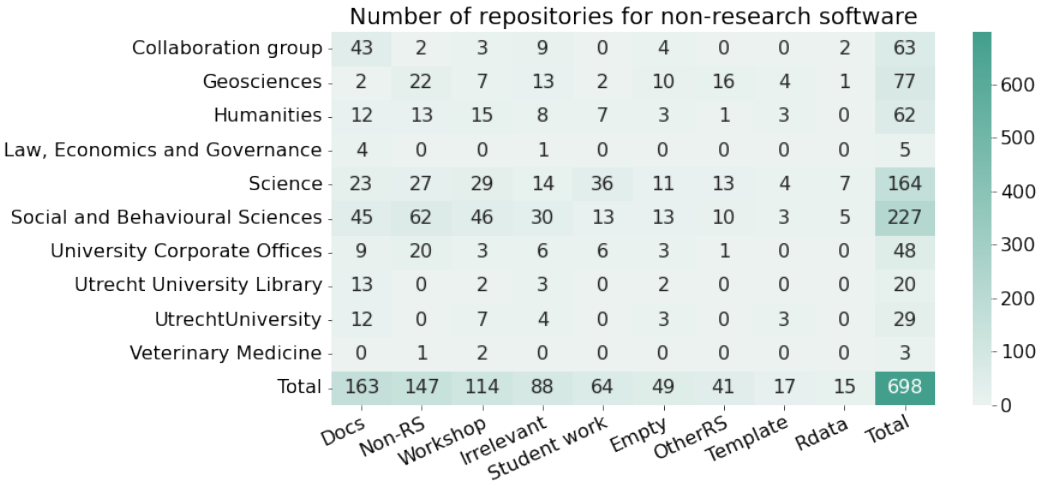


**Figure 4.5:** Number of non-research software repositories grouped by faculty and repository type.

low amount of relevant repositories. However, this does not necessarily mean that there is no existing research software within these faculties. It could be that it is hosted on different platforms, that their findability is lacking, or that our search method is not well-adjusted to these faculties. *Collaboration group* is also excluded from further analysis, as discussed in Chapter 3. *Social and Behavioral Sciences* will be shortened to *Social Sciences* for readability purposes in the following analysis.

The final processed dataset on which the following analysis will be done can be seen in Figure 4.6. All faculty groups now have a sufficient amount of research software repositories. The classes *Total RS* and *Total non-RS* are also relatively balanced, which is relevant for the classification.

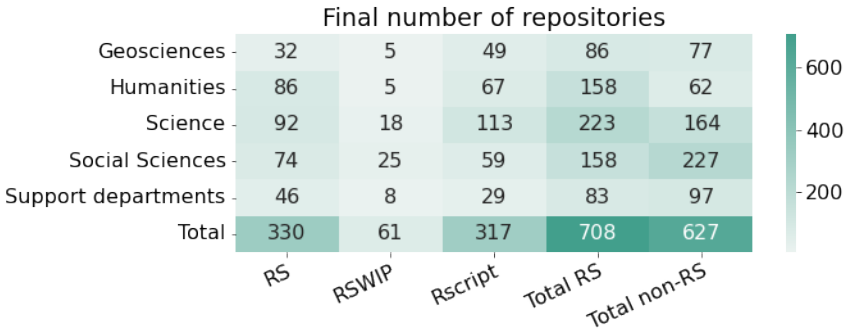


**Figure 4.6:** Number of final research software repositories grouped by faculty and repository type. *Total RS* aggregates the columns to the left, equivalent to previous Figures, while *total non-rs* additionally shows the number of repositories that are considered non-research software.

Figure 4.7 shows the histograms for all metrics split into the two classes *research software* and *non-research software*. We can see that the data are heavily long-tailed for both classes. There are a few repositories that have much higher metrics than most of the other ones. Except for number of contributors, research software usually has more large values than non-research software. Histograms with only research software and a heatmap of the most common topics can be seen in Appendix D.

Figure 4.8 shows the heatmaps for all averages of metrics collected for the repositories. This shows us that the averages are usually the highest for the repository type of research software. The number of languages and life span are roughly equal across the faculties. Geosciences have the highest average for open issues and the lowest average for contributors. Humanities have the lowest average for stargazers, forks, size, and topics often by a substantial amount. Science has the highest average for size and the lowest average for open issues. This is followed closely by Social Sciences, which are otherwise in-between other averages except for the number of languages. This might indicate that they primarily use only a few programming languages. The Support department has the highest average of stargazers by far, as well as forks, contributors, and topics.

Table 4.2 shows the highest occurring license for each faculty across the different repository types. There are a few things to note here: Science is the only faculty where Apache 2.0 appears, while every other faculty has a GPL license at the top. Each faculty has an MIT license at the top. Clearly, the dominating licenses are MIT and GPLv2/GPLv3.

| Faculty | RS | RSWIP | Rscript |
|---|---|---|---|
| Geosciences | GPLv3 | MIT | GPLv3, MIT |
| Humanities | GPLv2, MIT | Other | GPLv2 |
| Science | Apache 2.0 | MIT | MIT |
| Social Sciences | Other | GPLv3 | GPLv3, MIT |
| Support departments | MIT | GPLv3 | MIT |

**Table 4.2:** Highest occurring license per faculty per repository type.

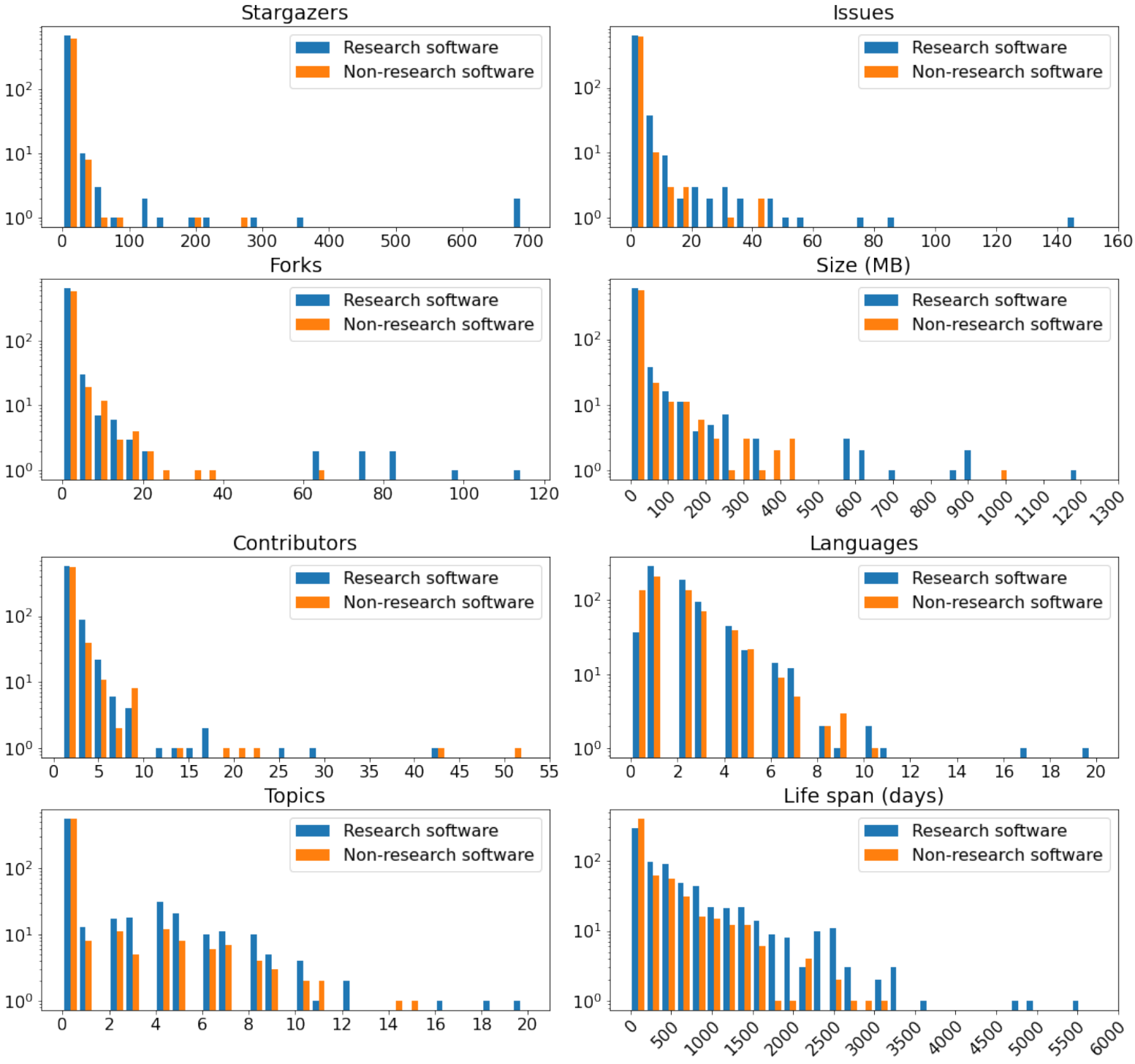


**Figure 4.7:** Histograms for metrics categorized by research software vs. non-research software. The y-axis is log-scaled.

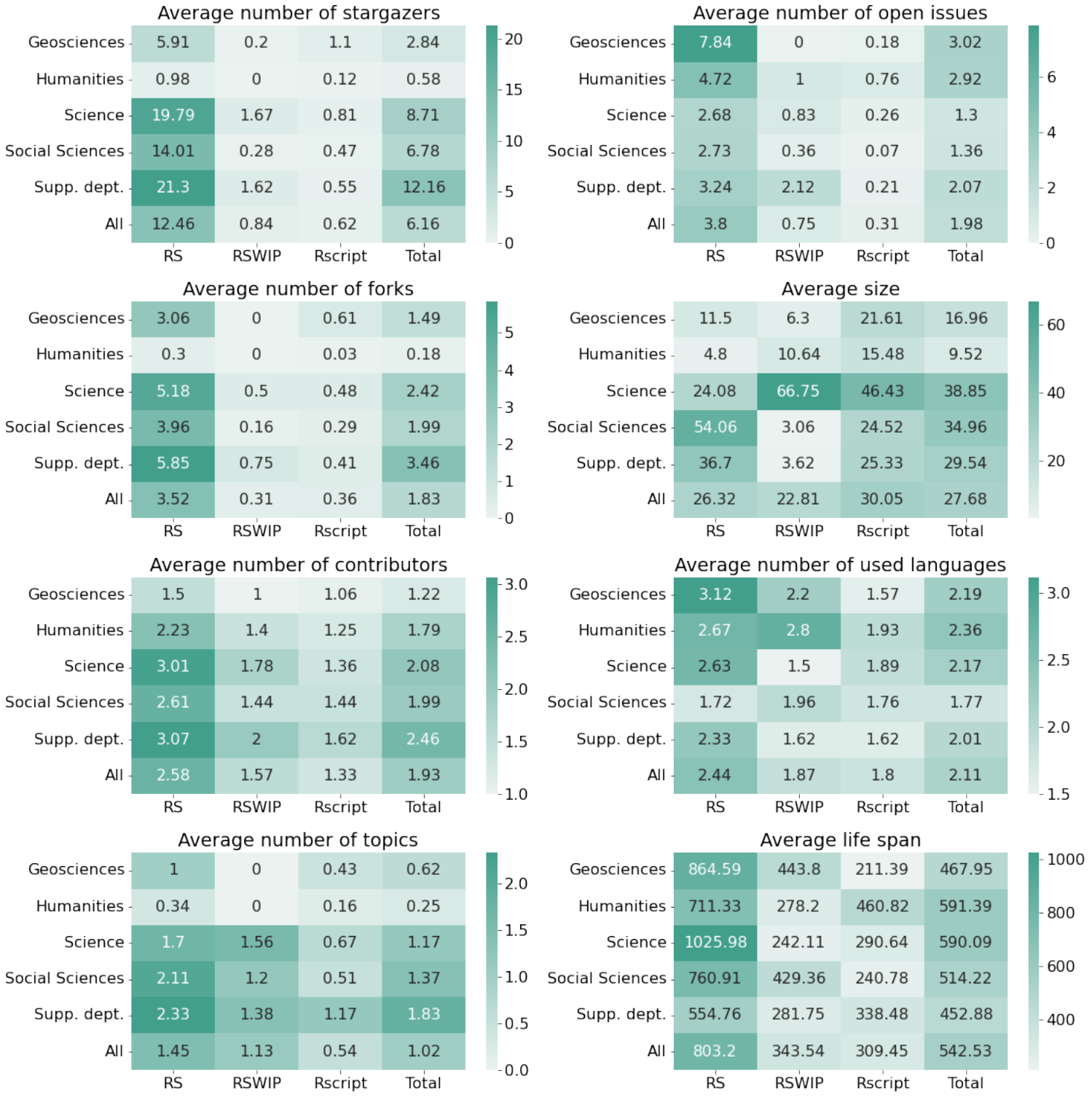


**Figure 4.8:** Heatmaps for all metrics grouped by faculty and repository type. *All* and *Total* refer to the average across all research software repositories. Therefore, these rows and columns do not equal the average across different rows and columns, since it also takes the number of repositories in each cell into account. This will also be the case for the following Figures.

We can take a deeper look into the license usage in Figure 4.9. This shows the license percentage for each faculty. What this also reveals is the percentage of no license usage. If we would also consider no license, this would be the dominating license usage for all faculties except the Support department. Most notable is Geosciences, with 57% of repositories having no license, which is 22 percentage points more than the following faculty, Humanities. Social Sciences have the least amount of unlicensed software. As we have seen from Table 4.2, the most common licenses are MIT and GPLv3. The licenses that follow afterwards are predominantly used by only one of the faculties.

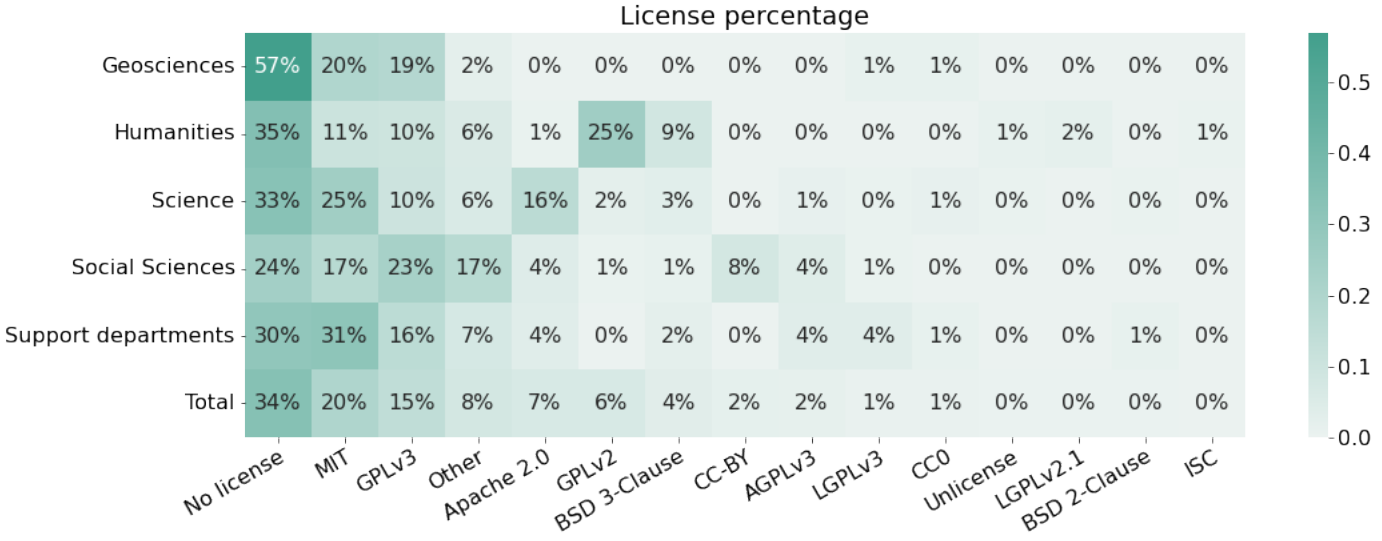


**Figure 4.9:** Licenses heatmap sorted from left to right by total percentage.

Table 4.3 shows the highest occurring language for each faculty across the different repository types. As expected, Python and R are dominating most categories. Science and Social Sciences stand out, with Python and R as the top language across all repository types, respectively. The only outliers to this are RSWIP and Rscript for Geosciences and Humanities. The category RSWIP for Geosciences and Humanities only have five repositories each, limiting the conclusions that can be drawn from this. That MATLAB would be common for Geosciences in Rscript is plausible, as it is widely used where matrix manipulation is necessary, like processing of images. Shell

is the most common language for Humanities in Rscript since the programming language Zep is often used here and is not detected by GitHub. This concerns many repositories from the Utrecht Institute of Linguistics OTS Lab (UiL OTS Lab).

| Faculty | RS | RSWIP | Rscript |
|---|---|---|---|
| Geosciences | Python | JavaScript, MATLAB, Python, Shell | MATLAB |
| Humanities | Python | JavaScript | Shell |
| Science | Python | Python | Python |
| Social Sciences | R | R | R |
| Support departments | Python | R | R |

**Table 4.3:** Highest occurring language per faculty per repository type.

Figure 4.10 shows all languages that were the top language in at least ten repositories. This confirms again that Social Sciences use mainly R for their research, while Python is most common for all other faculties. We can see a similar pattern to the licenses. After the first two top languages, the next ones are again predominantly used by only one of the faculties. The column *No language* is a special case where no language could be detected. In some cases, the contributors omitted the file extension, which GitHub uses to detect the languages.

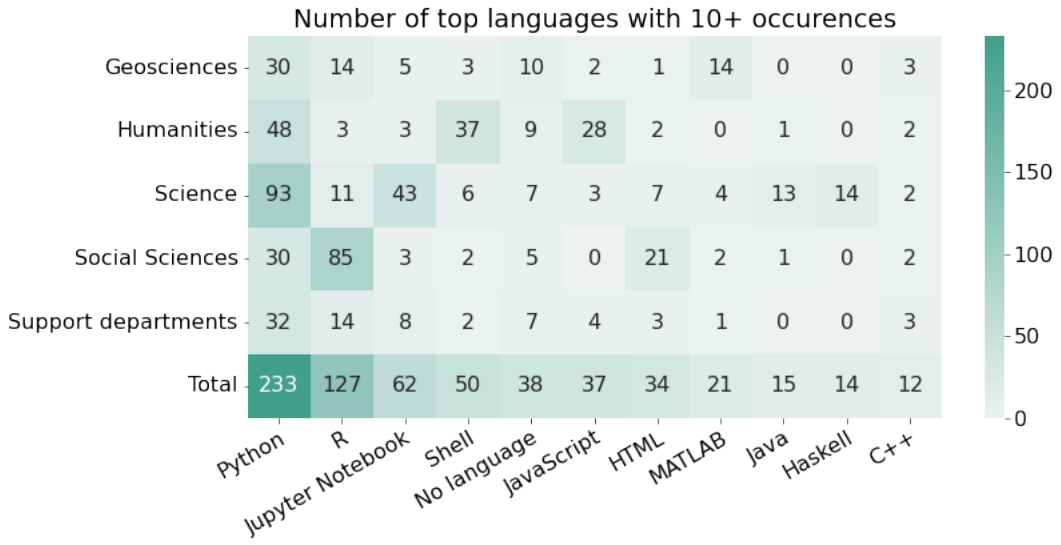


**Figure 4.10:** Languages heatmap sorted from left to right. Only languages with more than 10 total occurrences as top language are shown.

Figure 4.11 shows all languages that appeared in at least 10% of the repositories in a faculty. This considers all detected languages in a repository, not only the top language. While Jupyter Notebooks are often the dominant language, as Figure 4.10 shows, they are present in only a small amount of repositories. Shell also moved up by one spot; all faculties except for Social Sciences have used Shell in at least 20% of their repositories. This further indicates that Social Sciences might behave differently from other faculties regarding language usage. There is also a clear difference in language usage between Social Sciences and the other faculties, as R is used in 72% of repositories within that faculty, while the others barely use it. They have a strong preference for Python, which is present in about 50% of their repositories. Social Sciences, on the other hand, has Python present in only about 25% of the repositories.

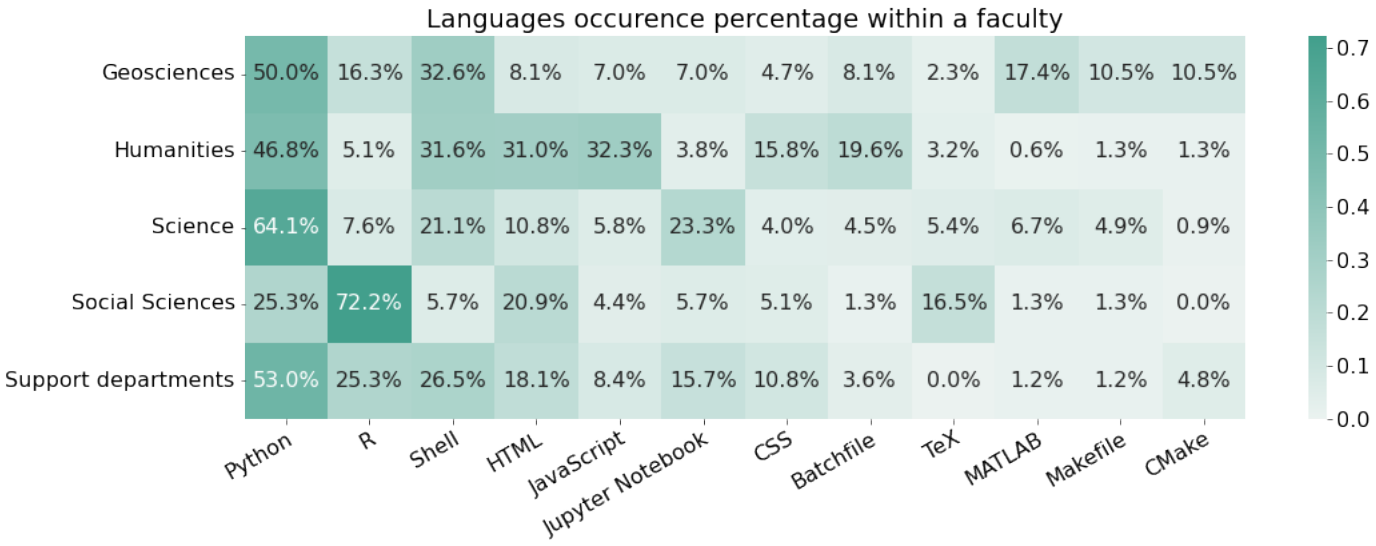


**Figure 4.11:** Languages heatmap sorted from left to right based on combined percentage per column. This shows all languages that occurred in at least 10% of the repositories within a faculty. The percentage represents how many repositories within a faculty used the mentioned language. As such, the percentage does not sum up to 100 across the columns or rows. If all languages within a faculty appeared in all repositories, they would all have 100%.

Figure 4.12 shows the correlation matrix for metrics. We can see a strong correlation between the number of stargazers and forks, between the number of forks and contributors, and a moderate relationship between the number of stargazers with contributors or topics. Forking is usually done to contribute to a repository, and users usually contribute to repositories they starred. A moderate relationship can be seen for number of issues with

contributors and languages and contributors with life span. This seems logical, as contributors often communicate via issues, and more contributors allow for a broader range of skills and used languages. They also facilitate software maintenance over longer periods.

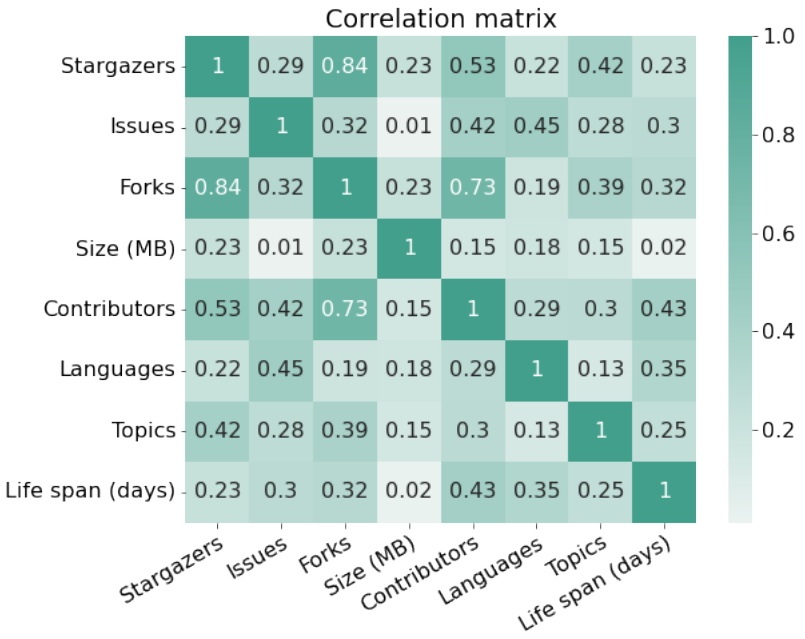


**Figure 4.12:** Correlation matrix for all metrics.

Descriptive statistics of all metrics can be seen in Table 4.4. The skewness and kurtosis show us that all metrics are heavily skewed and long-tailed, albeit to different degrees. The statistics per faculty can be seen in Appendix E. These show us that the median for all metrics except life span is relatively similar across all faculties.

| | Stargazers | Issues | Forks | Size (MB) | Contributors | Languages | Topics | Life span (days) |
|---|---|---|---|---|---|---|---|---|
| Minimum | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 | 0.00 | 0.00 | 0.00 |
| 25th percentile | 0.00 | 0.00 | 0.00 | 0.07 | 1.00 | 1.00 | 0.00 | 33.00 |
| Mean | 6.16 | 1.98 | 1.83 | 27.68 | 1.93 | 2.11 | 1.02 | 543.30 |
| Median | 0.00 | 0.00 | 0.00 | 0.68 | 1.00 | 2.00 | 0.00 | 297.00 |
| 75th percentile | 1.25 | 1.00 | 1.00 | 7.65 | 2.00 | 3.00 | 0.00 | 739.50 |
| Maximum | 699.00 | 148.00 | 116.00 | 1209.65 | 42.00 | 20.00 | 20.00 | 5609.00 |
| Skewness | 12.95 | 9.85 | 8.96 | 6.84 | 8.59 | 3.36 | 3.15 | 2.41 |
| Kurtosis | 189.27 | 127.60 | 87.08 | 55.84 | 101.50 | 21.87 | 12.73 | 8.27 |

**Table 4.4:** Descriptive statistics of numeric variables for all faculties

## 4.3 FAIR variables

Figure 4.13 shows the Jaccard similarity of all boolean FAIR variables. The Jaccard similarity between any variable with *repository open* approximately equals the percentage of true occurrences of the concerned variable since *repository open* is true for nearly all repositories. This shows that the three howfairis variables related to *registry, citation, and checklist* are found only in a small number of repositories, while the newly proposed ones are more frequent. We can also see that the correlation between the howfairis variables is not relatively high, except for the two variables related to registry and checklist. From the first and second rows, we can see that except for the variable *correct vcs usage* and *repository active*, all FAIR variables are more frequent or at least equal in case the repository has a license. *Is registered* has a higher similarity with *has contrib. guidelines*. Since research software that is registered is often developed by multiple people, it seems logical that these kinds of research software should have more guidelines for contribution. A similar logical argument can be made for the *has citation* and *version identifiable* since the Zenodo citation integration uses release tags. 268 out of 337 tags, or roughly 80% of repositories with tags, were correctly using identifiable versioning.

Figure 4.14 shows the heatmaps for the average percentage of each boolean FAIR variable per faculty. The upper plot shows the averages for research software, while the lower one shows averages for non-research software. We can immediately see from this that every FAIR variable has a higher percentage for research software than for non-research software. Large absolute increases can be seen in *has license, correct vcs usage, has install instructions, has tests, and version identifiable*. Social Sciences has the highest percentage of licenses across both classes for research software, while Geosciences has the lowest percentage. Social Sciences also have a high percentage of repositories that are registered, have citation enabled, or contain a checklist, compared to other faculties. Geosciences have the lowest average percentage of correct vcs usage, meaning that more than 20% of the research software repositories had only commits within a single day. Social Sciences and Support departments have a high

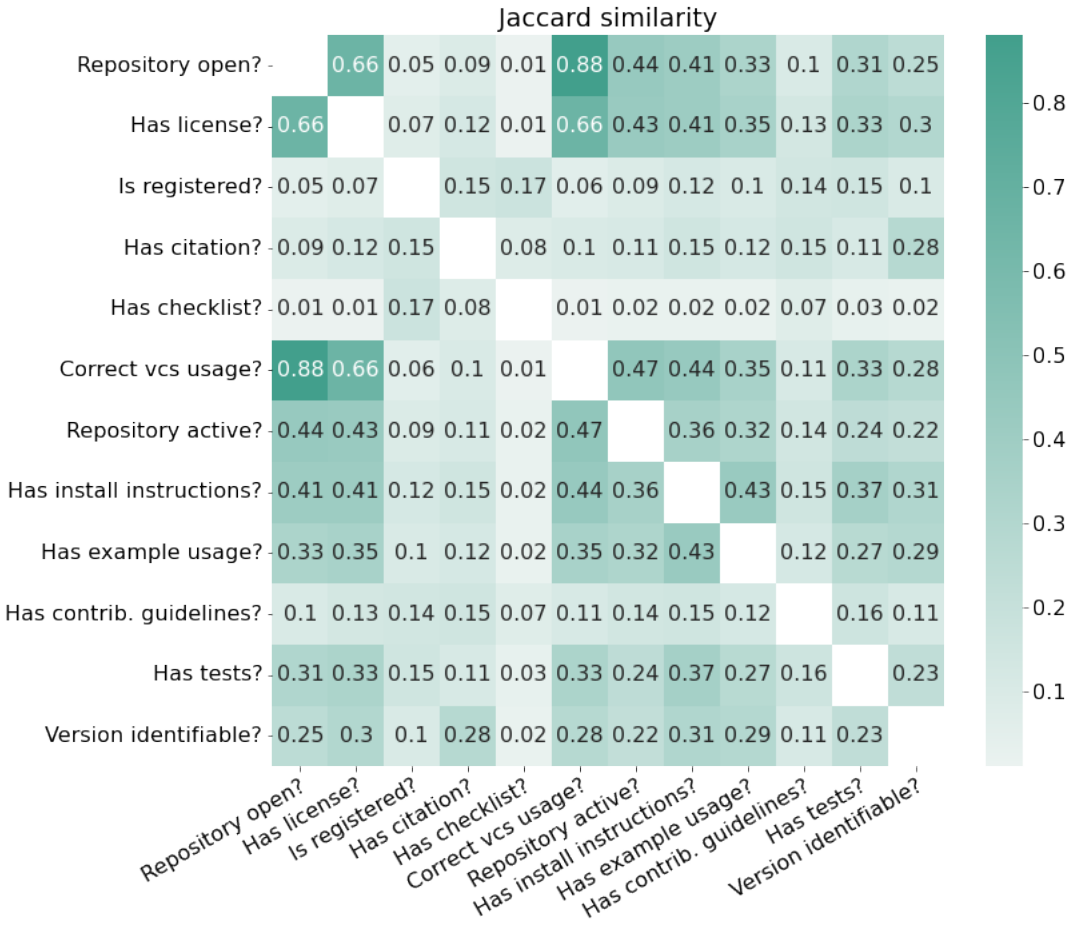


**Figure 4.13:** Jaccard similarity for all boolean FAIR variables.

percentage of active repositories, while other faculties are generally less active. Support departments, by far, have the highest percentage of install instructions and usage examples. For contribution guidelines, Social Sciences and Support departments again are around the same high percentage, while other faculties have only a minuscule amount. Geosciences and Humanities, the two faculties with the lowest percentage, are also the ones with the least amount of contributors. Interestingly, the Science faculty has the least amount of tests and identifiable versions on average.

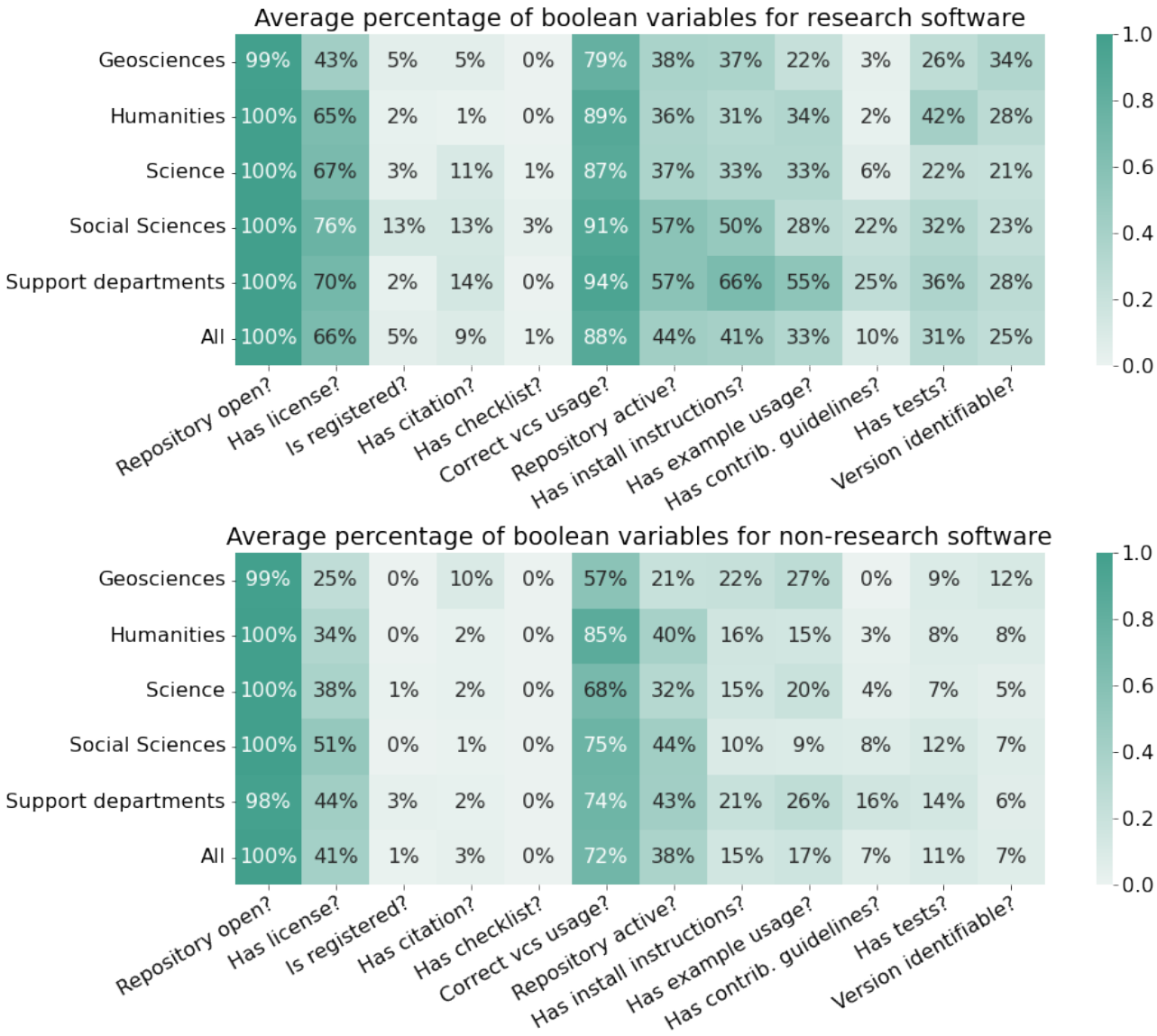


**Figure 4.14:** Heatmaps for all boolean FAIR variables grouped by faculty and repository type. The upper heatmap shows the averages for research software, while the lower one shows the averages for non-research software.

Figure 4.15 shows the FAIR score calculated as the sum of all boolean FAIR variables for each repository. We can see that *Rscripts* have a lower sum average than both *RS* and *RSWIP*. We can also see that Social Sciences and Support departments receive relatively high scores compared to the other faculties, with Geosciences having the lowest average score.

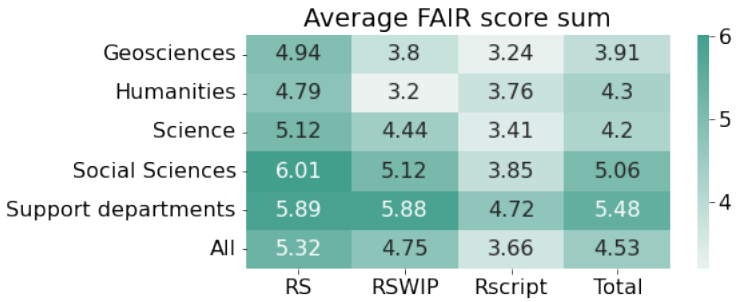


**Figure 4.15:** FAIR score of boolean FAIR variables grouped by faculty and repository type. The score for each repository is calculated as the number of true FAIR variables, which are all booleans.

## 4.4 Statistical tests

In order to statistically determine whether there are different characteristics in the metrics between faculties, we performed Kruskal-Wallis tests for each metric, followed by post hoc test analyses via Dunn's test. The results of the Kruskal-Wallis tests can be seen in Table 4.5. All metrics have significant differences except for life span. Figure 4.16 shows the post hoc test results. A p-value smaller than 0.05 indicates that the two groups come from a significantly different distribution. Overall, we can see that Science, Social Sciences and Support department are not too different, as they only have significant p-values within *open issues* and *topics*. Geosciences and Humanities, however, are frequently dissimilar to other faculties. Geosciences is the only different faculty for the number of contributors compared to all other ones.

| Variable | p-value | Significance |
|---|---|---|
| Stargazers | 1.247649e-08 | *** |
| Issues | 6.513571e-06 | *** |
| Forks | 1.301400e-09 | *** |
| Size (MB) | 3.112421e-04 | *** |
| Contributors | 3.651742e-05 | *** |
| Languages | 1.750939e-02 | * |
| Topics | 1.668033e-11 | *** |
| Life span (days) | 2.769494e-01 | |

**Table 4.5:** Kruskal-Wallis test results. Significance codes for p-values: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

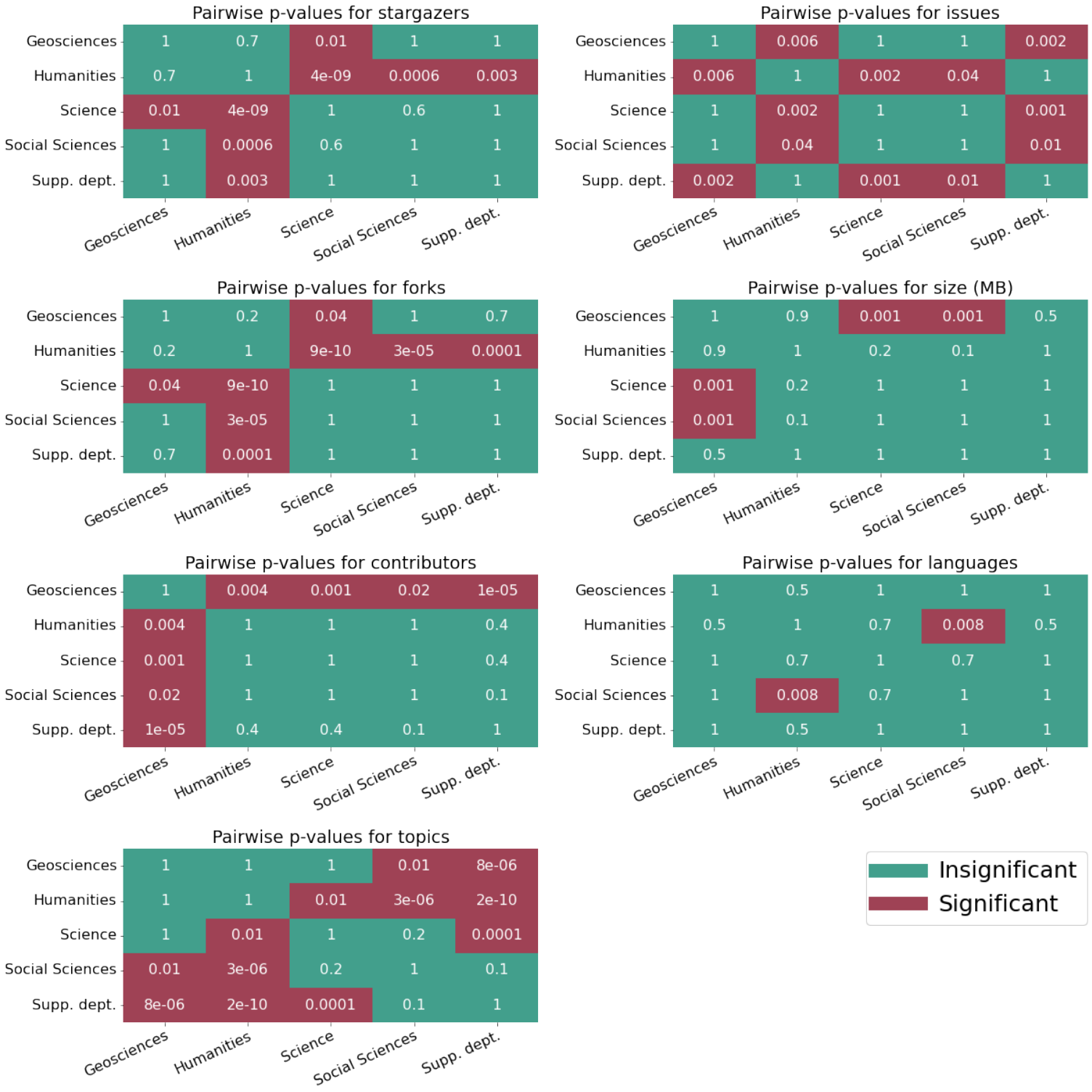


**Figure 4.16:** Dunn's test results for each variable.

## 4.5 Research software classification

We predict whether a repository is considered research software or not based on two trained models and a chance prediction. Figure 4.17 shows the confusion matrices for both model predictions and chance predictions for the research software classification. We can see from this that random forest performed better than logistic regression since there are fewer false negatives and false positives.

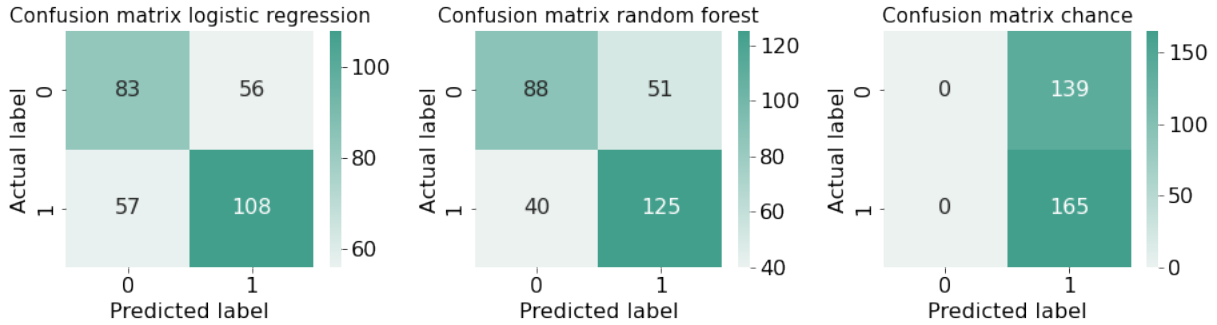


**Figure 4.17:** Confusion matrices for logistic regression, random forest, and chance.

Based on the confusion matrix, further performance measures are calculated in Figure 4.18, which shows the performance measures for both model and chance predictions. Both models outperform the chance classification in accuracy and precision. Random forest achieves a higher score in all performance measures than logistic regression and chance except for recall, which is expected. A random forest can better utilize complex variables than a logistic regression, where an increase in a variable value always increases or decreases the probability of a positive classification. This shows that the used variables improve classification compared to a chance classifier, that we can improve research software identification accuracy by 16 percentage points, and that a random forest is more suitable than logistic regression for this classification task.

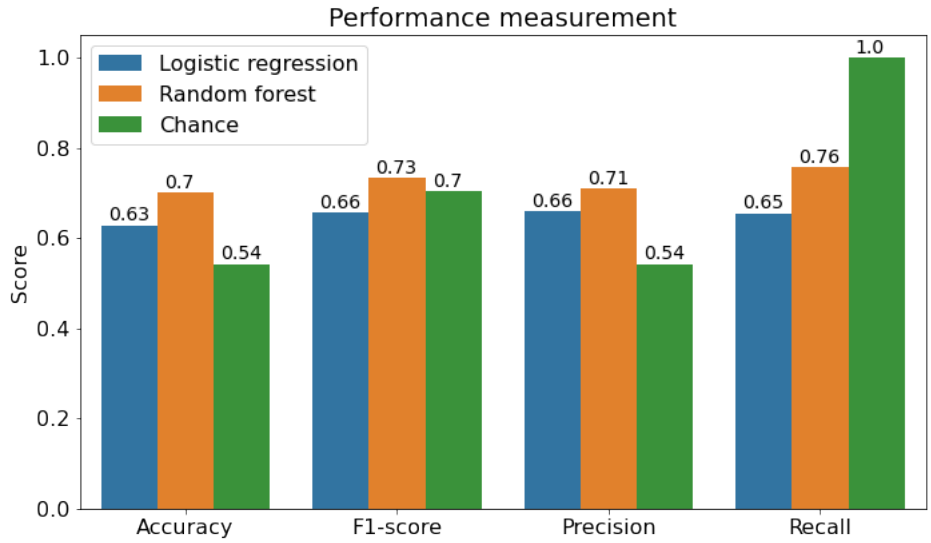


**Figure 4.18:** Performance measures of logistic regression, random forest, and chance.

Figure 4.19 shows the feature importance for the logistic regression model measured by the strength of the coefficients. The number of contributors has the highest effect, followed by life span and issues. After these, there is a steep decrease in the coefficient strength.

Figure 4.20 shows the permutation feature importance for the random forest model. Since the performance of this model was better than for logistic regression, we assume these feature importances to be more relevant. Life span is also an important feature for this model, with a 5.1% mean accuracy decrease, but the other top features differ. There is also a more gradual decrease in importance, as the variable languages follows with a 4.5% mean accuracy decrease. The following features *size, has install instructions, has license, has tests* all have a mean accuracy decrease of more than 2%.

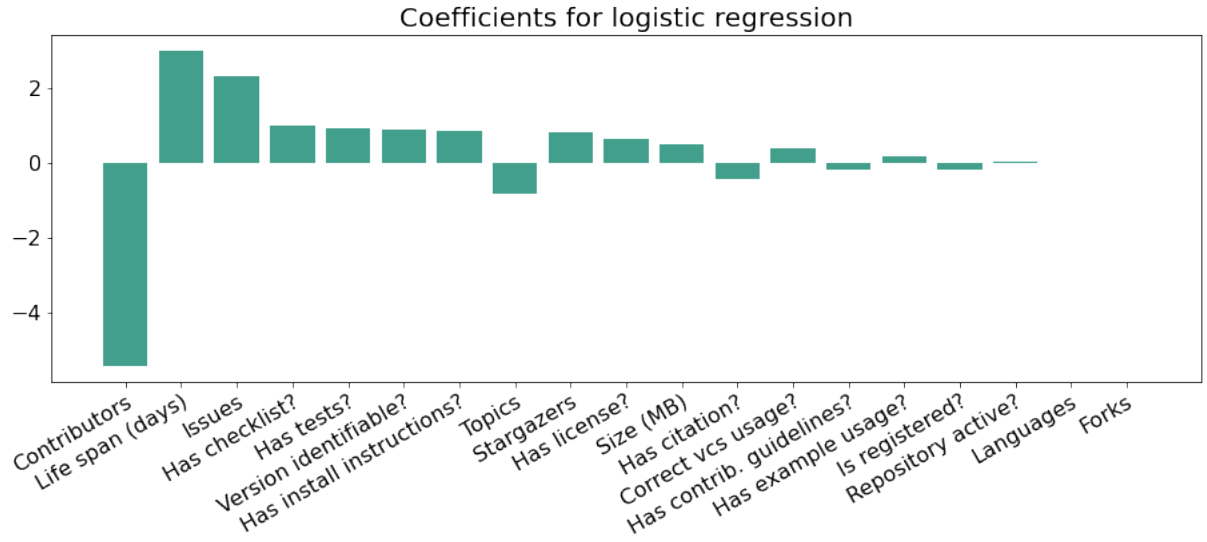


**Figure 4.19:** Feature importance for logistic regression shows the coefficients for each feature. A negative value indicates that an increase in that variable leads to a higher chance that the classified repository is not research software. The feature importance should therefore be viewed as an absolute value.

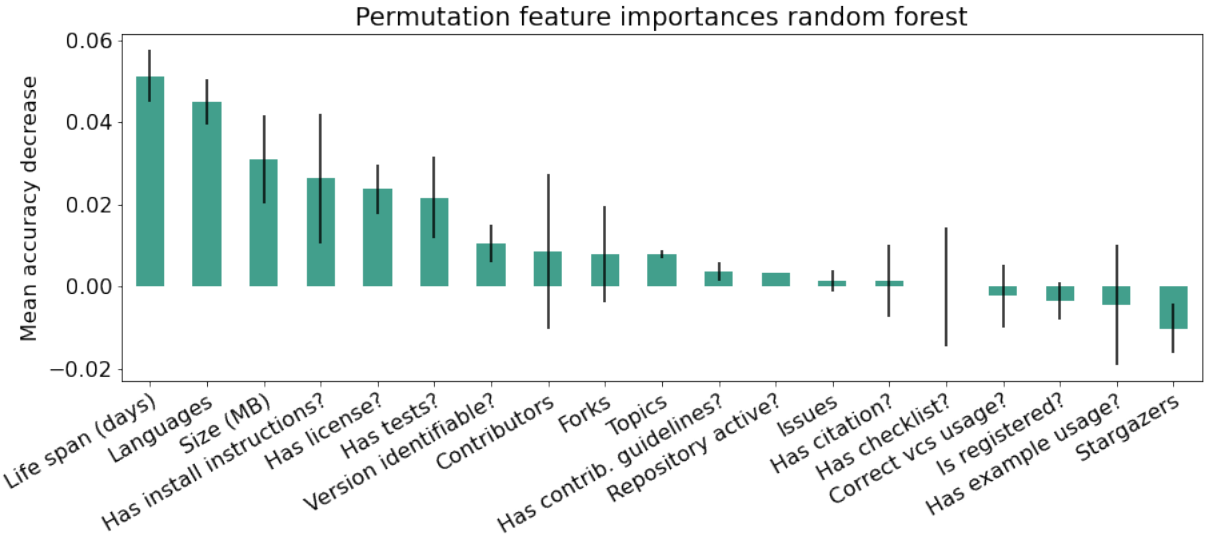


**Figure 4.20:** Feature importance for random forest. The y-axis shows by how many accuracy percentage points the random forest would perform worse if that feature would be removed. A negative importance can be interpreted that this is an irrelevant feature.

# 5 Discussion

We argue how the research questions are answered based on the results in Chapter 4. Subquestion 1 is related to the literature review and will not be further discussed. The FAIR variables that were retrieved from the literature review for subquestion 2 are validated in Section 5.1. Section 5.2 answers subquestion 3 and 4 that were related to subpopulation characteristics and support for application of FAIR variables. The last subquestion, which is about research software classification, is discussed in Section 5.3. Lastly, we look at the limitations of our study in Section 5.4.

## 5.1 Validation of additional FAIR variables

To validate **subquestion 2** (*how can FAIR principles be supplemented with additional variables?*), we looked at the Jaccard similarity coefficient of the FAIR variables in Figure 4.13 and percentages for research and non-research software in Figure 4.14. Since only a tiny percentage of repositories have information about registration, citation, and checklists, it is helpful to have additional measures of FAIRness. While all repositories should at least include license and citation information, it is not always sensible to register research software or scripts in a registry or to create a checklist. However, it can be argued that FAIR variables should be applicable for all kinds of research software. Unlike the other proposed FAIR variables, the commit-related variables *correct vcs usage* and *repository active* do not have a higher similarity with licensed repositories. This might indicate that these are not a good measure for FAIRness since licensing published software is a fundamental part of FAIRness. Additionally, while it might be a good practice in software development to frequently commit and correctly use version control, this aspect only relates marginally to FAIR principles. Their primary focus lies on measuring openness and sustainability. However, *correct vcs usage* might serve as a good measure to indicate which researchers might need support with the usage of software tools.

For the other FAIR variables, they relate to a stronger extent to the FAIR principles. Findability is improved through clear version identifiability, accessibility through contribution guidelines, reusability through install instructions, example usage, and tests. As such, we determine them as valuable additions as FAIR variables. It does not seem to be useful to determine faculty-specific FAIR variables. For example, even if Geosciences is significantly different to all other faculties regarding the number of contributors, it still makes sense to include contribution guidelines in case someone wants to contribute. Having such a guideline improves FAIRness, nonetheless.

## 5.2 Subpopulations and supporting application of FAIR variables

To answer **subquestion 3** (*are there different characteristics for different subpopulations in the data?*) and **subquestion 4** (*how can the application of FAIR variables for research software be supported?*), we looked at descriptive statistics, licenses, languages, topics, and statistical test analysis. We first address **subquestion 3**.

Figure 4.8 provided an overview of how available metrics differed between the faculties and repository types. Further faculty-specific differences could then be seen for both license and language usage. Social Sciences use R in more than 70% of their repositories. In contrast, other faculties use mainly Python to a lesser degree and a mix of other languages. Figure 4.14 revealed more differences regarding FAIR variables, with Figure 4.15 revealing a potential order of measured FAIRness across the faculties. This also empowers the argument that the RSE community is in a perfect position to support the improvement of FAIRness [62]. They are part of the *Support department*, which scored highest on average. It should be noted that the FAIR score from Figure 4.15 is currently designed such that each FAIR variable has an equal weight. We have already discussed the importance of having a license, which should therefore be weighted higher than having a checklist. Pico et al. [88] implemented such a weighting for their analysis. The statistical tests confirmed that all metrics except life span are significantly

different. This is particularly interesting since Hasselbring et al. [63] found the life span median to be vastly different across the two subpopulations they compared. However, they only presented the median value without a statistical test for significance. The tests also showed that Geosciences are significantly different to all other faculties regarding contributors, which means that they usually have less expertise available, possibly leading to the lowest FAIR score. It might also be worthwhile to consider the further levels of subpopulations. We looked only at the level of faculties. While it is possible to go down further levels, it might also be of interest to group the employees via their job position, which is also available information. How does the research software published by a full professor differ from that of a research engineer or PhD student? Are the effects larger than across faculties? This allows us to understand whether position-based training or materials might be more appropriate than faculty-based training. Based on these findings, we conclude that there are different characteristics at the faculty level.

This finding becomes relevant in answering **subquestion 4**. In order to support the application of FAIR variables, we need to consider the FAIRness maturity level of the researchers. As we have seen, this seems to vary across the faculties. We can specifically examine Geosciences in different regards. Looking at *licenses* in Figure 4.9, we see that most repositories in this faculty had no license, while other faculties performed considerably better in that regard. Figure 4.14 also showed that Geosciences has a lower percentage of correct vcs usage. As we have previously discussed regarding subquestion 2, this variable relates to good practices in software engineering. As such, this might indicate that Geosciences require more fundamental training in this topic and hands-on usage of GitHub. Such training should still be open for anybody to participate in but could be more geared towards the academic needs of Geosciences. A document or video-based tutorial would possibly also be worthwhile to consider. However, while this may save resources, it also comes with drawbacks. A training for this purpose seems most suited as it allows inexperienced researchers to ask questions and receive appropriate feedback. Understanding the tools that are used during the process of creating research software is a prerequisite to creating

FAIR research software.

Coming back to the licenses, there would ideally be no repository without a license since unlicensed software can not be legally reused for any purpose. Therefore, this topic should be made more aware across all faculties as it concerns them equally. However, it is also noteworthy that research-related repositories have more licenses on average compared to non-research-related repositories and previously conducted analyses such as Russell et al. [94]. While the significance of other FAIR-related aspects is less extreme, they follow a similar logic in being relevant for all faculties. For the FAIR variables in this dataset, this concerns *has citation, has install instructions, has example usage, has contribution guidelines, has tests, version identifiable*. We see varying percentages across the variables and faculties, ranging from 1% to 66%. As such, these are aspects that can be improved everywhere. This could be accomplished by faculty-agnostic training or a document regarding best FAIR practices. A document has the advantage of being easily editable, such that new additions or changes in best FAIR practices can be incorporated with little effort to avoid the provision of outdated information. In addition, a yearly report on FAIRness that analyzes changes per faculty, similar to this study, might be helpful in tracking the effect of the implemented measures. It might also incentivize researchers to improve FAIRness, as such a report should provide transparent criteria on how to achieve this. This has been a big hurdle in improving FAIRness for many researchers, as stated in our findings from SWORDS@UU consultations and presentations in Section 2.4. Such a yearly report also becomes feasible since this study provides a labelled dataset and code for the analysis. If the phases are repeated, only new repositories need to be labelled, which is a minor effort compared to having no previously labelled data.

## 5.3 Identifying research software

Section 4.5 answers subquestion 5 (*how well can we identify research software with available data?*). We used logistic regression and random forest to predict the class of the repositories. Our results showed that both models outperform the majority class prediction. Since random forest performs better than logistic regression across all performance measures, we assume this model is better suited for our purposes, dataset size, and available variables. Using such a classification model can be useful for future tasks. Relating back to the yearly report, it would first be necessary to label newly collected repositories each time. Using the model for prediction as a first automated labelling step reduces manual labor and will help in making such a yearly report even more feasible. The manual labelling process is time-consuming and error-prone. It should therefore be supported by automated solutions as much as possible.

The used features for our models were all numeric or boolean variables. Further development on classifier models should consider including categorical variables, such as the given license or most used language. Also, it would be beneficial for future classification to develop more variables. One example would be to include the readme text as n-grams or other potential forms. The classification could additionally be done for each subpopulation. However, considering the amount of currently available repositories, we decided against this. This might be more suitable if future work can also incorporate previous university employees.

## 5.4 Limitations of study

This study only considers current researchers due to the API restrictions of the employee pages. If there were public access to previous employees, it would be possible to widen this study to include all previous employees. Another limitation is the sole focus on GitHub data. While it is the most popular platform, UU also hosts its own

GitLab server with active repositories. However, the highest number of stars for a repository is seven. This might indicate that most of the relevant research software is indeed found on other platforms. Furthermore, the variable retrieval implementation can be improved to be more accurate. For example, the variable *has citation* does not take into account if there is citation information in the readme, which is quite common. The variable *has install instructions* only checks for mentions of the *install* keyword in the readme, but could also take language-specific metadata into account. This includes *setup.py, DESCRIPTION, pom.xml, and package.json* files for Python, R, Java, and JacaScript, respectively. Incorporating these files would also allow us to collect information on dependencies and versions, as Lamprecht et al. [79] described.

# 6 Conclusion

This study aimed to identify how open source publications on GitHub can be used to infer actionable recommendations for RSE practice to improve the research software landscape of an organization. In order to do so, we reviewed the FAIR principles, identified suitable variables to measure FAIRness, and conducted an exploratory data analysis. The quantitative analysis was applied to UU to determine different characteristics in the faculties, support for the application of FAIR variables, and how well research software can be identified.

Our method retrieved 176 users that had 1521 repositories. 823 of the repositories can be considered research software. We found that proposed FAIR variables are a helpful addition to measuring FAIRness and that there are different characteristics in the faculties, indicating a need for different possibilities of support for applying FAIR variables. Among other findings, we found out that Geosciences have 57% of unlicensed software, while the next highest percentage is much lower with 35% for the Humanities. There is also a clear difference in language usage between Social Sciences, who primarily use R, and the other faculties, who primarily use Python. We additionally provided first models for classifying research software to facilitate future research software identification, achieving an accuracy of 70%. The FAIR data analysis for GitHub allows UU to make data-based decisions, as a first analysis of the research software landscape at UU and a first labelled dataset for reuse were provided. This has not only confirmed and refuted beliefs in existing literature, but also provided novel findings and laid the foundation for repeated analyses of this kind. Additionally, the SWORDS@UU framework was extended with additional validated FAIR variables.

There are several topics for future work, which are explained in detail in Chapter 5. Data collection can be improved in several ways, analysis can be extended and applied to more variations of subpopulations, and classification of research software can be refined. We conclude that the conducted analysis allows us to infer actionable recommendations for RSE practice and encourage others to reuse and improve the method.

# References


[1] FAIR self assessment tool - ARDC, . URL https://ardc.edu.au/resources/aboutdata/fair-data/fair-self-assessment-tool/. (accessed 2022-05-31).

[2] Draft Software FAIR assessment template, . URL https://docs.google.com/spreadsheets/d/1_6olVmarOJp1nt7VpUHraicsTEpNsp_1-histomaNkg/edit. (accessed 2022-05-31).

[3] FAIR4RS Roadmap Metrics WG charter. URL https://docs.google.com/document/d/1BpzecVx4ZvSNfHD-UHhofZVdA6qiP_ENrmozmiq9zY4/edit. (accessed 2022-05-31).

[4] Members – FAIR 4 Research Software (FAIR4RS) Working Group – FORCE11. URL https://force11.org/groups/fair-4-research-software-fair4rs-working-group/. (accessed 2022-08-08).

[5] GitHub REST API. URL https://ghdocs-prod.azurewebsites.net/en/rest. (accessed 2022-05-25).

[6] BadgeApp, . URL https://bestpractices.coreinfrastructure.org/en. (accessed 2022-10-05).

[7] BinderHub, . URL https://binderhub.readthedocs.io/en/latest/. (accessed 2022-03-22).

[8] BioConductor, . URL https://www.bioconductor.org/. (accessed 2022-04-05).

[9] FAIR4RS Roadmap Metrics WG charter, . URL https://docs.google.com/document/d/1BpzecVx4ZvSNfHD-UHhofZVdA6qiP_ENrmozmiq9zY4/edit?usp=embed_facebook. (accessed 2022-05-20).

[10] FAIR Research Software, . URL https://fair-software.nl/home. (accessed 2022-04-27).

[11] ghapi documentation reference, . URL https://ghapi.fast.ai/. (accessed 2022-05-25).

[12] GitHub. Making Your Code Citable., . URL https://guides.github.com/activities/citable-code/. (accessed 2022-05-13).

[13] GO Build Charter - GO FAIR US, . URL https://docs.google.com/document/d/1SNTcjQb_TDMuqc-kov0vNvcaOm6ZKYQT9-9qEe-_-6g/edit. (accessed 2022-05-31).

[14] Journal of Open Research Software, . URL http://openresearchsoftware.metajnl.com/. (accessed 2022-03-22).

[15] Journal of Open Source Software, . URL https://joss.theoj.org. (accessed 2022-03-22).

[16] Opening the archive, one API (and one FOSDEM) at a time, . URL https://www.softwareheritage.org/2017/02/04/archive-api/. (accessed 2022-04-05).

[17] Papers with Code - The latest in Machine Learning, . URL https://paperswithcode.com/. (accessed 2022-05-23).

[18] Pure | The world's leading RIMS or CRIS | Elsevier Solutions, . URL https://www.elsevier.com/solutions/pure. (accessed 2022-05-23).

[19] Six Recommendations for implementation of FAIR practice by the FAIR in practice task force of the European open science cloud FAIR working group - Publications Office of the EU, . URL https://op.europa.eu/en/publication-detail/-/publication/4630fa57-1348-11eb-9a54-01aa75ed71a1/language-en. (accessed 2022-04-24).

[20] SoBigData.eu, . URL http://sobigdata.eu/index. (accessed 2022-03-22).

[21] Software Management Plans, . URL https://www.software.ac.uk/software-management-plans. (accessed 2022-05-19).

[22] About TNO. URL https://www.tno.nl/en/about-tno/. (accessed 2022-08-11).

[23] Medewerkers - Universiteit Utrecht. URL https://www.uu.nl/staff/Search. (accessed 2022-08-11).

[24] IEEE Standard Glossary of Software Engineering Terminology. *IEEE Std 610.12-1990*, pages 1–84, December 1990. doi:10.1109/IEEESTD.1990.101064. Conference Name: IEEE Std 610.12-1990.

[25] FAIR4Software reading materials, February 2020. URL https://www.rd-alliance.org/group/software-source-code-ig/wiki/fair4software-reading-materials. (accessed 2022-05-14).

[26] Codemeta Generator, May 2021. URL https://github.com/codemeta/codemeta-generator. (accessed 2022-04-27).

[27] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, 2016. URL https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf.

[28] Anton Akhmerov, Maria Cruz, Niels Drost, Cees Hof, Tomas Knapen, Mateusz Kuzak, Carlos Martinez-Ortiz, and Yasemin Turkyilmaz-van der Velden. Making Research Software a First-Class Citizen in Research, April 2019. URL https://zenodo.org/record/2647436.

[29] Olusegun Akinwande, H.G Dikko, and Samson Agboola. Variance Inflation Factor: As a Condition for the Inclusion of Suppressor Variable(s) in Regression Analysis. *Open Journal of Statistics*, 05:754–767, January 2015. doi:10.4236/ojs.2015.57075.

[30] Alexis. Answer to "Bonferroni correction on multiple Kruskal-Wallis tests", January 2015. URL https://stats.stackexchange.com/a/133449/261994. (accessed 2022-08-24).

[31] The Research Software Alliance. Task forces, July 2020. URL https://www.researchsoft.org/taskforces/. (accessed 2022-08-08).

[32] David B. Allison, Andrew W. Brown, Brandon J. George, and Kathryn A. Kaiser. Reproducibility: A tragedy of errors. *Nature*, 530(7588):27–29, February 2016. ISSN 1476-4687. doi:10.1038/530027a. URL http://www.nature.com/articles/530027a. Number: 7588 Publisher: Nature Publishing Group.

[33] Marti J. Anderson. A new method for non-parametric multivariate analysis of variance. *Austral Ecology*, 26(1):32–46, 2001. ISSN 1442-9993. doi:10.1111/j.1442-9993.2001.01070.pp.x. URL http://onlinelibrary.wiley.com/doi/abs/10.1111/j.1442-9993.2001.01070.pp.x. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1442-9993.2001.01070.pp.x.

[34] Hartwig Anzt, Felix Bach, Stephan Druskat, Frank Löffler, Axel Loewe, Bernhard Y. Renard, Gunnar Seemann, Alexander Struck, Elke Achhammer, Piush Aggarwal, Franziska Appel, Michael Bader, Lutz Brusch, Christian

Busse, Gerasimos Chourdakis, Piotr Wojciech Dabrowski, Peter Ebert, Bernd Flemisch, Sven Friedl, Bernadette Fritzsch, Maximilian D. Funk, Volker Gast, Florian Goth, Jean-Noël Grad, Jan Hegewald, Sibylle Hermann, Florian Hohmann, Stephan Janosch, Dominik Kutra, Jan Linxweiler, Thilo Muth, Wolfgang Peters-Kottig, Fabian Rack, Fabian H.C. Raters, Stephan Rave, Guido Reina, Malte Reißig, Timo Ropinski, Joerg Schaarschmidt, Heidi Seibold, Jan P. Thiele, Benjamin Uekermann, Stefan Unger, and Rudolf Weeber. An environment for sustainable research software in Germany and beyond: current state, open challenges, and call for action. *F1000Research*, 9: 295, January 2021. ISSN 2046-1402. doi:10.12688/f1000research.23224.2. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7845155/.

[35] Michelle Barker. FAIR4RS Roadmap Report. February 2022. doi:10.5281/zenodo.6239373. URL https://zenodo.org/record/6239373. Publisher: Zenodo.

[36] Fabien C. Y. Benureau and Nicolas P. Rougier. Re-run, Repeat, Reproduce, Reuse, Replicate: Transforming Code into Scientific Contributions. *Frontiers in Neuroinformatics*, 11, 2018. ISSN 1662-5196. URL https://www.frontiersin.org/article/10.3389/fninf.2017.00069.

[37] Francine Berman and Merce Crosas. The Research Data Alliance: Benefits and Challenges of Building a Community Organization. *Harvard Data Science Review*, 2(1), jan 31 2020. https://hdsr.mitpress.mit.edu/pub/i4e009fo.

[38] Kelly Blincoe, Jyoti Sheoran, Sean Goggins, Eva Petakovic, and Daniela Damian. Understanding the popular users: Following, affiliation influence and leadership on GitHub. *Information and Software Technology*, 70:30–39, February 2016. ISSN 0950-5849. doi:10.1016/j.infsof.2015.10.002. URL https://www.sciencedirect.com/science/article/pii/S0950584915001688.

[39] Martin Boeckhout, Gerhard A. Zielhuis, and Annelien L. Bredenoord. The FAIR guiding principles for data stewardship: fair enough? *European Journal of Human Genetics*, 26(7):931–936, July 2018. ISSN 1018-4813, 1476-5438. doi:10.1038/s41431-018-0160-0. URL http://www.nature.com/articles/s41431-018-0160-0.

[40] Hudson Borges, Andre Hora, and Marco Tulio Valente. Understanding the Factors that Impact the Popularity of GitHub Repositories. *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages

334–344, October 2016. doi:10.1109/ICSME.2016.31. URL http://arxiv.org/abs/1606.04984. arXiv: 1606.04984.

[41] Christine L. Borgman. *Big Data, Little Data, No Data: Scholarship in the Networked World*. MIT Press, Cambridge, MA, USA, January 2015. ISBN 978-0-262-02856-1.

[42] Scott Chacon and Ben Straub. *Pro git*. Apress, 2014.

[43] Neil Chue Hong. Making Software A First-Class Citizen, September 2019. URL https://figshare.com/articles/presentation/Making_Software_A_First-Class_Citizen/9862835/1. Publisher: figshare.

[44] Neil Chue Hong. FAIR4RS Software (FAIR4RS), March 2022. URL https://zenodo.org/record/6374314.

[45] Neil P. Chue Hong, Daniel S Katz, Michelle Barker, Anna-Lena Lamprecht, Carlos Martinez, Fotis E Psomopoulos, Jen Harrow, Leyla Jael Castro, Morane Gruenpeter, Paula Andrea Martinez, Tom Honeyman, Alexander Struck, Allen Lee, Axel Loewe, Ben van Werkhoven, Catherine Jones, Daniel Garijo, Esther Plomp, Francoise Genova, Hugh Shanahan, Joanna Leng, Maggie Hellström, Manodeep Sinha, Mateusz Kuzak, Patricia Herterich, Qian Zhang, Sharif Islam, Susanna-Assunta Sansone, Tom Pollard, Udayanto Dwi Atmojo, Alan Williams, Andreas Czerniak, Anna Niehues, Anne Claire Fouilloux, Bala Desinghu, Céline Richard, Charles Gray, Chris Erdmann, Daniel Nüst, Daniele Tartarini, Hartwig Anzt, Ilian Todorov, James McNally, Javier Moldon, Jessica Burnett, Khalid Belhajjame, Laurents Sesink, Lorraine Hwang, Marcos Roberto, Mark D Wilkinson, Mathieu Servillat, Matthias Liffers, Merc Fox, Nick Lynch, Paula Martinez Lavanchy, Sandra Gesing, Sarah Stevens, Martinez Cuesta, Silvio Peroni, Stian Soiland-Reyes, Tom Bakker, Tovo Rabemanantsoa, Vanessa Sochat, and Yo Yehudi. FAIR Principles for Research Software (FAIR4RS Principles). page 32, 2021. doi:10.15497/RDA00068.

[46] Neil P. Chue Hong, Daniel S. Katz, Michelle Barker, Anna-Lena Lamprecht, Carlos Martinez, Fotis E. Psomopoulos, Jen Harrow, Leyla Jael Castro, Morane Gruenpeter, Paula Andrea Martinez, and Tom Honeyman. FAIR Principles for Research Software (FAIR4RS Principles). March 2022. doi:10.15497/RDA00068. URL https://rd-alliance.org/group/fair-research-software-fair4rs-wg/outcomes/fair-principles-research-software-fair4rs-0. Publisher: Research Data Alliance.

[47] Neil Philippe Chue Hong. What makes software FAIR? Reviewing the Towards FAIR Principles for Research Software paper and new research related to FAIR software: A report from FAIR4RS Subgroup 4. Technical report, Zenodo, June 2021. URL https://zenodo.org/record/4908919.

[48] Valerio Cosentino, Javier L. Cánovas Izquierdo, and Jordi Cabot. A Systematic Mapping Study of Software Development With GitHub. *IEEE Access*, 5:7173–7192, 2017. ISSN 2169-3536. doi:10.1109/ACCESS.2017.2682323. Conference Name: IEEE Access.

[49] Jonathan de Bruin, Keven Quach, Christopher Slewe, and Anna-Lena Lamprecht. Scan and revieW of Open Research Data and Software at Utrecht University, 9 2021. URL https://github.com/UtrechtUniversity/SWORDS-UU.

[50] Roberto Di Cosmo and Stefano Zacchiroli. Software Heritage: Why and How to Preserve Software Source Code. In *iPRES 2017 - 14th International Conference on Digital Preservation*, pages 1–10, Kyoto, Japan, September 2017. URL https://hal.archives-ouvertes.fr/hal-01590958.

[51] Directorate-General for Research and Innovation (European Commission). *Turning FAIR into reality: final report and action plan from the European Commission expert group on FAIR data*. Publications Office of the European Union, LU, 2018. ISBN 978-92-79-96546-3. URL https://data.europa.eu/doi/10.2777/1524.

[52] Stephan Druskat, Jurriaan H. Spaaks, Neil Chue Hong, Robert Haines, James Baker, Spencer Bliven, Egon Willighagen, David Pérez-Suárez, and Alexander Konovalov. Citation File Format. August 2021. doi:10.5281/zenodo.5171937. URL https://zenodo.org/record/5171937. Publisher: Zenodo.

[53] European Organization For Nuclear Research and OpenAIRE. Zenodo, 2013. URL https://www.zenodo.org/.

[54] Benedikt Fecher and Sascha Friesike. Open Science: One Term, Five Schools of Thought. In Sönke Bartling and Sascha Friesike, editors, *Opening Science: The Evolving Guide on How the Internet is Changing Research, Collaboration and Scholarly Publishing*, pages 17–47. Springer International Publishing, Cham, 2014. ISBN 978-3-319-00026-8. doi:10.1007/978-3-319-00026-8_2. URL https://doi.org/10.1007/978-3-319-00026-8_2.

[55] Leyla Garcia, Bérénice Batut, Melissa L. Burke, Mateusz Kuzak, Fotis Psomopoulos, Ricardo Arcila, Teresa K. Attwood, Niall Beard, Denise Carvalho-Silva, Alexandros C. Dimopoulos, Victoria Dominguez del Angel, Michel Dumontier, Kim T. Gurwitz, Roland Krause, Peter McQuilton, Loredana Le Pera, Sarah L. Morgan, Päivi Rauste, Allegra Via, Pascal Kahlem, Gabriella Rustici, Celia W. G. van Gelder, and Patricia M. Palagi. Ten simple rules for making training materials FAIR. *PLOS Computational Biology*, 16(5):e1007854, May 2020. ISSN 1553-7358. doi:10.1371/journal.pcbi.1007854. URL https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1007854. Publisher: Public Library of Science.

[56] Leyla Jael Garcia Castro, Michelle Barker, Neil Chue Hong, Fotis Psomopoulos, Jennifer Harrow, Daniel S. Katz, Mateusz Kuzak, Paula Andrea Martinez, and Allegra Via. Software as a first-class citizen in research. 2020. doi:10.4126/FRL01-006423290. URL https://repository.publisso.de/resource/frl:6423290. Medium: application/pdf Publisher: PUBLISSO.

[57] Mohammad Gharehyazie, Baishakhi Ray, and Vladimir Filkov. Some from Here, Some from There: Cross-Project Code Reuse in GitHub. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 291–301, May 2017. doi:10.1109/MSR.2017.15.

[58] Morane Gruenpeter. Software as a first class output in a FAIR ecosystem, October 2021. URL https://zenodo.org/record/5563028.

[59] Morane Gruenpeter, Roberto Di Cosmo, Hylke Koers, Patricia Herterich, Rob Hooft, Jessica Parland-von Essen, Jonas Tana, Tero Aalto, and Sarah Jones. M2.15 Assessment report on 'FAIRness of software'. October 2020. doi:10.5281/zenodo.4095092. URL https://zenodo.org/record/4095092. Publisher: Zenodo.

[60] Morane Gruenpeter, Daniel S. Katz, Anna-Lena Lamprecht, Tom Honeyman, Daniel Garijo, Alexander Struck, Anna Niehues, Paula Andrea Martinez, Leyla Jael Castro, Tovo Rabemanantsoa, Neil P. Chue Hong, Carlos Martinez-Ortiz, Laurents Sesink, Matthias Liffers, Anne Claire Fouilloux, Chris Erdmann, Silvio Peroni, Paula Martinez Lavanchy, Ilian Todorov, and Manodeep Sinha. Defining Research Software: a controversial discussion. Technical report, Zenodo, September 2021. URL https://zenodo.org/record/5504016.

[61] Gruenpeter, Morane, Di Cosmo, Roberto, Koers, Hylke, Herterich, Patricia, Hooft, Rob, Parland-von Essen, Jessica, Tana, Jonas, Aalto, Tero, and Jones, Sarah. M2.15 Assessment report on 'FAIRness of software'. October 2020. doi:10.5281/ZENODO.4095092. URL https://zenodo.org/record/4095092.

[62] Wilhelm Hasselbring, Leslie Carr, Simon Hettrick, Heather Packer, and Thanassis Tiropanis. From FAIR research data toward FAIR and open research software. *it - Information Technology*, 62(1):39–47, February 2020. ISSN 2196-7032. doi:10.1515/itit-2019-0040. URL http://www.degruyter.com/document/doi/10.1515/itit-2019-0040/html. Publisher: De Gruyter Oldenbourg.

[63] Wilhelm Hasselbring, Leslie Carr, Simon Hettrick, Heather Packer, and Thanassis Tiropanis. Open Source Research Software. *Computer*, 53(8):84–88, August 2020. ISSN 1558-0814. doi:10.1109/MC.2020.2998235. Conference Name: Computer.

[64] Sandra Heiler. Semantic interoperability. *ACM Computing Surveys*, 27(2):271–273, June 1995. ISSN 0360-0300. doi:10.1145/210376.210392. URL https://doi.org/10.1145/210376.210392.

[65] Tom Honeyman, Qian Zhang, Christian Pagé, Francoise Genova, Neil P. Chue Hong, Paula Andrea Martinez, Michelle Barker, Catherine Jones, Leyla Garcia-Castro, Morane Gruenpeter, and Carlos Martinez. Subgroup 7: Governance. URL https://docs.google.com/document/d/1pBBs8hSF8m3WsFRFjJuFHd1sC-jfPV3_0xUTMVzEuMc/edit?usp=embed_facebook. (accessed 2022-05-20).

[66] Yan Hu, Jun Zhang, Xiaomei Bai, Shuo Yu, and Zhuo Yang. Influence analysis of Github repositories. *SpringerPlus*, 5(1):1268, August 2016. ISSN 2193-1801. doi:10.1186/s40064-016-2897-7. URL https://doi.org/10.1186/s40064-016-2897-7.

[67] Carl Huberty and John Morris. Multivariate Analysis Versus Multiple Univariate Analyses. *Psychological Bulletin*, 105:302–308, March 1989. doi:10.1037/0033-2909.105.2.302.

[68] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. German, and Daniela Damian. The promises and perils of mining GitHub. In *Proceedings of the 11th Working Conference on Mining Software Repositories - MSR 2014*, pages 92–101, Hyderabad, India, 2014. ACM Press. ISBN 978-1-4503-2863-0. doi:10.1145/2597073.2597074. URL http://dl.acm.org/citation.cfm?doid=2597073.2597074.

[69] B. M. Katz and M. McSweeney. A Multivariate Kruskal-Wallis Test With Post Hoc Procedures. *Multivariate Behavioral Research*, 15(3):281–297, July 1980. ISSN 0027-3171. doi:10.1207/s15327906mbr1503_4.

[70] Daniel S. Katz, Michelle Barker, Paula Andrea Martinez, Hartwig Anzt, Alejandra Gonzalez-Beltran, and Tom Bakker. The Research Software Alliance (ReSA) and the community landscape, March 2020. URL https://zenodo.org/record/3699950. Type: dataset.

[71] Daniel S. Katz, Michelle Barker, Neil P. Chue Hong, Leyla Jael Castro, and Paula Andrea Martinez. The FAIR4RS team: Working together to make research software FAIR, June 2021. URL https://zenodo.org/record/5037157. Conference Name: 2021 Collegeville Workshop on Scientific Software - Software Teams (Collegeville2021) Publisher: Zenodo.

[72] Daniel S. Katz, Neil P. Chue Hong, Michelle Barker, and Morane Gruenpeter. FAIR4RS WG subgroup community consultation March 2021. March 2021. doi:10.5281/zenodo.4635410. URL https://zenodo.org/record/4635410. Publisher: Zenodo.

[73] Daniel S. Katz, Morane Gruenpeter, Tom Honeyman, Lorraine Hwang, Mark D. Wilkinson, Vanessa Sochat, Hartwig Anzt, Carole Goble, and for FAIR4RS Subgroup 1. A Fresh Look at FAIR for Research Software. *arXiv:2101.10883 [cs]*, February 2021. URL http://arxiv.org/abs/2101.10883. arXiv: 2101.10883.

[74] Daniel S. Katz, Fotis Psomopoulos, and Leyla Jael Castro. Working Towards Understanding the Role of FAIR for Machine Learning. 2021. doi:10.4126/FRL01-006429415. URL https://repository.publisso.de/resource/frl:6429415. Medium: application/pdf Publisher: PUBLISSO.

[75] Daniel S. Katz, Michelle Barker, Neil P. Chue Hong, Leyla Jael Garcia-Castro, Morane Gruenpeter, Jennifer Harrow, Carlos Martinez, Paula Andrea Martinez, and Fotis E. Psomopoulos. The Overlap Between FAIR for Research Software and Open Science, March 2022. URL https://zenodo.org/record/6340732.

[76] Pavneet Singh Kochhar, Tegawendé F. Bissyandé, David Lo, and Lingxiao Jiang. An Empirical Study of Adoption of Software Testing in Open Source Projects. In *2013 13th International Conference on Quality Software*, pages 103–112, July 2013. doi:10.1109/QSIC.2013.57. ISSN: 2332-662X.

[77] Sven Kosub. A note on the triangle inequality for the Jaccard distance, December 2016. URL http://arxiv.org/abs/1612.02696. arXiv:1612.02696 [cs, stat].

[78] Mateusz Kuzak, Michelle Barker, Robin Richardson, Jessica Burnett, Carole Goble, and Leyla Garcia. FAIR4RS subgroup 2 - summary report. URL https://docs.google.com/document/d/1zPjeJgVKg4q1nEYTxRJIas2w3MYYlUVOnjUtJRLp7QI/edit?usp=embed_facebook. (accessed 2022-05-03).

[79] Anna-Lena Lamprecht, Leyla Garcia, Mateusz Kuzak, Carlos Martinez, Ricardo Arcila, Eva Martin Del Pico, Victoria Dominguez Del Angel, Stephanie van de Sandt, Jon Ison, Paula Andrea Martinez, Peter McQuilton, Alfonso Valencia, Jennifer Harrow, Fotis Psomopoulos, Josep Ll. Gelpi, Neil Chue Hong, Carole Goble, and Salvador Capella-Gutierrez. Towards FAIR principles for research software. *Data Science*, 3(1):37–59, June 2020. ISSN 24518492, 24518484. doi:10.3233/DS-190026. URL https://www.medra.org/servlet/aliasResolver?alias=iospress&doi=10.3233/DS-190026.

[80] Wanwangying Ma, Lin Chen, Yuming Zhou, and Baowen Xu. What Are the Dominant Projects in the GitHub Python Ecosystem? In *2016 Third International Conference on Trustworthy Systems and their Applications (TSA)*, pages 87–95, September 2016. doi:10.1109/TSA.2016.23.

[81] Paula Andrea Martinez, Alexander Struck, Leyla Jael Castro, Daniel Garijo, Axel Loewe, Sandra Gesing, Michelle Barker, Neil Chue Hong, Christopher Erdmann, Carlos Martinez-Ortiz, and Susanna-Assunta Sansone. A Survey on Adoption Guidelines for the FAIR4RS Principles: Dataset, March 2022. URL https://zenodo.org/record/6375540. Type: dataset.

[82] Carlos Martinez-Ortiz, Mateusz Kuzak, Jurriaan H. Spaaks, Jason Maassen, and Tom Bakker. Five recommendations for "FAIR software". December 2020. doi:10.5281/zenodo.4310217. URL https://zenodo.org/record/4310217. Publisher: Zenodo.

[83] Carlos Martinez-Ortiz, Daniel S. Katz, Anna-Lena Lamprecht, Michelle Barker, Axel Loewe, Anne Fouilloux, Jane Wyngaard, Daniel Garijo, Javier Moldon, Leyla Jael Castro, Daniel Wheeler, Joost Rutger Demian Albers, and Allen

Lee. FAIR4RS: Adoption support. February 2022. doi:10.5281/zenodo.6258366. URL https://zenodo.org/record/6258366. Publisher: Zenodo.

[84] Eva Martín Del Pico, Salvador Capella-Gutierrez, and Josep Lluis Gelpi. Automating the monitoring of research software FAIR metrics. *F1000Research*, 9, June 2020. doi:10.7490/f1000research.1117992.1. URL https://f1000research.com/posters/9-555.

[85] Barend Mons, Cameron Neylon, Jan Velterop, Michel Dumontier, Luiz Olavo Bonino da Silva Santos, and Mark D. Wilkinson. Cloudy, increasingly FAIR; revisiting the FAIR Data guiding principles for the European Open Science Cloud. *Information Services & Use*, 37(1):49–56, March 2017. ISSN 0167-5265. doi:10.3233/ISU-170824. URL https://content-iospress-com.proxy.library.uu.nl/articles/information-services-and-use/isu824. Publisher: IOS Press.

[86] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[87] Limor Peer, Florio Arguillas, Tom Honeyman, Nadica Miljković, and Karsten Peters-von Gehlen. Challenges of Curating for Reproducible and FAIR Research Output. page 26, July 2021.

[88] Eva Martín del Pico, Josep Lluís Gelpí, and Salvador Capella-Gutiérrez. FAIRsoft - A practical implementation of FAIR principles for research software. Technical report, bioRxiv, May 2022. URL https://www.biorxiv.org/content/10.1101/2022.05.04.490563v1.

[89] Tom Preston-Werner. Semantic versioning 2.0.0. *línea]. Available: http://semver.org*, 2013.

[90] Keven Quach. Mapping-Research-Software-Landscapes-through-Exploratory-Studies-of-GitHub-Data, 5 2022. URL https://github.com/kequach/Thesis-Mapping-RS.

[91] Baishakhi Ray, Daryl Posnett, Vladimir Filkov, and Premkumar Devanbu. A large scale study of programming languages and code quality in github. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, pages 155–165, New York, NY, USA, November 2014. Association

for Computing Machinery. ISBN 978-1-4503-3056-5. doi:10.1145/2635868.2635922. URL https://doi.org/10.1145/2635868.2635922.

[92] Research Data Alliance/FORCE11 Software Source Code Identification WG, Alice Allen, Anita Bandrowski, Peter Chan, Roberto Di Cosmo, Martin Fenner, Leyla Garcia, Morane Gruenpeter, Catherine M. Jones, Daniel S. Katz, John Kunze, Moritz Schubotz, and Ilian T. Todorov. Use cases and identifier schemes for persistent software source code identification (V1.0). 2020. doi:10.15497/RDA00053. URL https://zenodo.org/record/4312464#.X9BuzGgza70.

[93] J. Ringersma and M. Miedema. Do I-PASS for FAIR? Measuring the FAIR-ness of Research Organizations. *Data Science Journal*, 20(1):30, October 2021. ISSN 1683-1470. doi:10.5334/dsj-2021-030. URL https://datascience.codata.org/article/10.5334/dsj-2021-030/.

[94] Pamela H. Russell, Rachel L. Johnson, Shreyas Ananthan, Benjamin Harnke, and Nichole E. Carlson. A large-scale analysis of bioinformatics code on GitHub. *PLOS ONE*, 13(10):e0205898, October 2018. ISSN 1932-6203. doi:10.1371/journal.pone.0205898. URL https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0205898. Publisher: Public Library of Science.

[95] Jyoti Sheoran, Kelly Blincoe, Eirini Kalliamvakou, Daniela Damian, and Jordan Ell. Understanding "watchers" on GitHub. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 336–339, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 978-1-4503-2863-0. doi:10.1145/2597073.2597114. URL https://doi.org/10.1145/2597073.2597114.

[96] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The Hadoop Distributed File System. In *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–10, Incline Village, NV, USA, May 2010. IEEE. ISBN 978-1-4244-7152-2. doi:10.1109/MSST.2010.5496972. URL http://ieeexplore.ieee.org/document/5496972/.

[97] Jurriaan H. Spaaks, Mateusz Kuzak, Carlos Martinez-Ortiz, Ben van Werkhoven, Edidiong Etuk, Shyam Saladi, Andrew Holding, Erik Tjong Kim Sang, Faruk Diblen, and Stefan Verhoeven. howfairis. URL https://github.com/fair-software/howfairis.

[98] Jurriaan H. Spaaks, Tom Klaver, Stefan Verhoeven, Faruk Diblen, Jason Maassen, Erik Tjong Kim Sang, Pushpanjali Pawar, Christiaan Meijer, Lars Ridder, Lode Kulik, Tom Bakker, Vincent van Hees, Laurens Bogaardt, Adriënne Mendrik, Bram van Es, Jisk Attema, Willem van Hage, Elena Ranguelova, Rob van Nieuwpoort, Ronny Gey, and Hoskins Zach. Research Software Directory, December 2020. URL https://zenodo.org/record/4311332.

[99] John W. Tukey. *Exploratory Data Analysis*. Addison-Wesley, 1977.

[100] Caspar J. Van Lissa, Andreas M. Brandmaier, Loek Brinkman, Anna-Lena Lamprecht, Aaron Peikert, Marijn E. Struiksma, and Barbara M. I. Vreede. WORCS: A workflow for open reproducible code in science. *Data Science*, 4 (1):29–49, January 2021. ISSN 2451-8484. doi:10.3233/DS-210031. URL https://content-iospress-com.proxy.library.uu.nl/articles/data-science/ds210031. Publisher: IOS Press.

[101] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009. ISBN 1441412697.

[102] CURE-FAIR WG. Cure-fair wg: Case statement, October 2020. URL https://www.rd-alliance.org/sites/default/files/case_statement/CURE_FAIR_WG_CaseStatement_9October2020v2.pdf. (accessed 2022-05-31).

[103] FAIR4RS WG. FAIR4RS Subgroup 4 - reading list of new research, February 2021. URL https://zenodo.org/record/4555865. Type: dataset.

[104] Mark D. Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E. Bourne, Jildau Bouwman, Anthony J. Brookes, Tim Clark, Mercè Crosas, Ingrid Dillo, Olivier Dumon, Scott Edmunds, Chris T. Evelo, Richard Finkers, Alejandra Gonzalez-Beltran, Alasdair J. G. Gray, Paul Groth, Carole Goble, Jeffrey S. Grethe, Jaap Heringa, Peter A. C. 't Hoen, Rob Hooft, Tobias Kuhn, Ruben Kok, Joost Kok, Scott J. Lusher, Maryann E. Martone, Albert Mons, Abel L. Packer, Bengt Persson, Philippe Rocca-Serra, Marco Roos, Rene van Schaik, Susanna-Assunta Sansone, Erik Schultes, Thierry Sengstag, Ted Slater, George Strawn, Morris A. Swertz, Mark Thompson, Johan van der Lei, Erik van Mulligen, Jan Velterop, Andra Waagmeester, Peter Wittenburg, Katherine Wolstencroft, Jun Zhao, and Barend Mons. The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data*, 3(1):

160018, March 2016. ISSN 2052-4463. doi:10.1038/sdata.2016.18. URL http://www.nature.com/articles/sdata201618. Number: 1 Publisher: Nature Publishing Group.

[105] Claes Wohlin. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering - EASE '14*, pages 1–10, London, England, United Kingdom, 2014. ACM Press. ISBN 978-1-4503-2476-2. doi:10.1145/2601248.2601268. URL http://dl.acm.org/citation.cfm?doid=2601248.2601268.

[106] Matthew Wolf, Jeremy Logan, Kshitij Mehta, Daniel Jacobson, Mikaela Cashman, Angelica M. Walker, Greg Eisenhauer, Patrick Widener, and Ashley Cliff. Reusability First: Toward FAIR Workflows. In *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 444–455, September 2021. doi:10.1109/Cluster48925.2021.00053. ISSN: 2168-9253.

[107] Lou Woodley and Katie Pratt. The CSCCE Community Participation Model – A framework to describe member engagement and information flow in STEM communities. August 2020. doi:10.5281/zenodo.3997802. URL https://zenodo.org/record/3997802. Publisher: Zenodo.

# Acronyms

**ANOVA** Analysis of variance.

**ARDC** Australian Research Data Commons.

**CI/CD** continuous integration and continuous delivery or continuous deployment.

**CSCCE** Center for Scientific Collaboration and Community Engagement.

**DOI** Digital Object Identifier.

**EOSC** European Open Science Cloud.

**FAIR** findable, accessible, interoperable, and reusable.

**FAIR4RS** FAIR for Research Software.

**FAIR4RS WG** FAIR for Research Software Working Group.

**FORCE11** Future Of Research Communications and E-Scholarship.

**FOSS** Free and/or Open Source Software.

**PID** Persistent Unique Identifier.

**RDA** Research Data Alliance.

**ReSA** Research Software Alliance.

**RMSE** root-mean-square error.

**RSE** Research Software Engineer.

**SMP** Software Management Plan.

**SWORDS@UU** Scan and revieW of Open Research Data and Software at Utrecht University.

**UiL OTS Lab** Utrecht Institute of Linguistics OTS Lab.

**UMCU** University Medical Center Utrecht.

**UU** Utrecht University.

**VIF** variance inflation factor.

# A FAIR principles comparison

| FAIR Guiding Principles (2016) | Towards FAIR Principles for research software (2020) | FAIR4RS Principles (2022) |
|---|---|---|
| Findable | | |
| The first step in (re)using data is to find them. Metadata and data should be easy to find for both humans and computers. Machine-readable metadata are essential for automatic discovery of datasets and services, so this is an essential component of the FAIRification process. | The main concern of findability for research software is to ensure software can be identified unambiguously when looking for it using common search strategies. | Software, and its associated metadata, is easy for both humans and machines to find. |
| F1. (Meta)data are assigned a globally unique and persistent identifier | F1. Software and its associated metadata have a global, unique and persistent identifier for each released version. | F1. Software is assigned a globally unique and persistent identifier.<br>F1.1. Components of the software representing levels of granularity are assigned distinct identifiers.<br>F1.2. Different versions of the software are assigned distinct identifiers. |
| F2. Data are described with rich metadata (defined by R1 below) | F2. Software is described with rich metadata. | F2. Software is described with rich metadata. |
| F3. Metadata clearly and explicitly include the identifier of the data they describe | F3. Metadata clearly and explicitly include identifiers for all the versions of the software it describes. | F3. Metadata clearly and explicitly include the identifier of the software they describe. |
| F4. (Meta)data are registered or indexed in a searchable resource | F4. Software and its associated metadata are included in a searchable software registry. | F4. Metadata are FAIR, searchable and indexable. |
| A. Accessible | | |
| Once the user finds the required data, she/he needs to know how can they be accessed, possibly including authentication and authorisation. | Accessibility translates into retrievability [...] however, we found mere retrievability not enough. In order for anyone to use any research software, a working version of the software needs to be available. | Software, and its metadata, is retrievable via standardized protocols. |
| A1. (Meta)data are retrievable by their identifier using a standardized communications protocol | A1. Software and its associated metadata are accessible by their identifier using a standardized communications protocol. | A1. Software is retrievable by its identifier using a standardized communications protocol. |
| A1.1. The protocol is open, free, and universally implementable | A1.1. The protocol is open, free, and universally implementable. | A1.1. The protocol is open, free, and universally implementable. |

| | | |
|---|---|---|
| A1.2. The protocol allows for an authentication and authorization procedure, where necessary | A1.2. The protocol allows for an authentication and authorization procedure, where necessary. | A1.2. The protocol allows for an authentication and authorization procedure, where necessary. |
| A2. Metadata are accessible, even when the data are no longer available | A2. Software metadata are accessible, even when the software is no longer available. | A2. Metadata are accessible, even when the software is no longer available. |
| I. Interoperable | | |
| The data usually needs to be integrated with other data. In addition, the data need to interoperate with applications or workflows for analysis, storage, and processing. | Interoperability for research software can be understood in two dimensions: as part of workflows (horizontal dimension) and as stack of digital objects that need to work together at compilation and execution times (vertical dimension) | Software interoperates with other software by exchanging data and/or metadata, and/or through interaction via application programming interfaces (APIs), described through standards. |
| I1. (Meta)data use a formal, accessible, shared, and broadly applicable language for knowledge representation. | I1. Software and its associated metadata use a formal, accessible, shared and broadly applicable language to facilitate machine readability and data exchange. | I1. Software reads, writes and exchanges data in a way that meets domain-relevant community standards |
| I2. (Meta)data use vocabularies that follow FAIR principles | I2.1. Software and its associated metadata are formally described using controlled vocabularies that follow the FAIR principles.<br>I2.2. Software use and produce data in types and formats that are formally described using controlled vocabularies that follow the FAIR principles. | Now split between F4 and I1. |
| I3. (Meta)data include qualified references to other (meta)data | | I2. Software includes qualified references to other objects. |
| | I4S. Software dependencies are documented and mechanisms to access them exist. | |
| R. Reusable | | |
| The ultimate goal of FAIR is to optimize the reuse of data. To achieve this, metadata and data should be well-described so that they can be replicated and/or combined in different settings. | Reusability in the context of software has many dimensions. At its core, reusability aims for someone to be able to reuse software reproducibly. | Software is both usable (can be executed) and reusable (can be understood, modified, built upon, or incorporated into other software). |
| R1. (Meta)data are richly described with a plurality of accurate and relevant attributes | R1. Software and its associated metadata are richly described with a plurality of accurate and relevant attributes. | R1. Software is described with a plurality of accurate and relevant attributes. |

| | | |
|---|---|---|
| R1.1. (Meta)data are released with a clear and accessible data usage license | R1.1. Software and its associated metadata have independent, clear and accessible usage licenses compatible with the software dependencies. | R1.1. Software is given a clear and accessible license. |
| R1.2. (Meta)data are associated with detailed provenance | R1.2. Software metadata include detailed provenance, detail level should be community agreed. | R1.2. Software is associated with detailed provenance. |
| R1.3. (Meta)data meet domain-relevant community standards | R1.3. Software metadata and documentation meet domain-relevant community standards. | R3. Software meets domain-relevant community standards. |
| | | R2. Software includes qualified references to other software. |

# B Repository type labels

| Label | Description | Research software? |
|---|---|---|
| Research Software | Is the tool used for research or was it produced during research? If it is installable (has setup file, is a package, Docker installation) it counts as Research Software. GUIs and (shiny) apps also count as Research Software. If there is no documentation but code structure implies reusability, it is still counted as Research Software, even if the author describes it as a "simple script". Parts used by other software that are reusable also count as Research Software. | Yes |
| Rscript | Generally things that are only for reproducibility of specific analyses or workflows, not reusability directly. Scripts that need (heavy) modification to be used for other purposes that the original intent. Additionally, scripts for benchmarking, testing or showcasing of research software. Queries like SPARQL scripts are also considered research scripts. | Yes |
| RSWIP | Research Software that is currently WIP or in alpha/experimental phase. Based on available information within a repository or corresponding documentation. Incomplete abandoned software is also considered RSWIP. Releases in R with version number less than 1.0 considered RSWIP according to best practices: https://lifecycle.r-lib.org/articles/stages.html#experimental | Yes |
| Empty | Includes placeholder repositories that have not much more than a few lines of text or a license. | No |
| Non-RS | Other kind of software, for example for simplifying or automating workflows that were not clearly developed during research process. Software and scripts created for learning or other purposes not directly related to research. | No |

| | | |
|---|---|---|
| Rdata | Repositories with only research data. If they contain research software or scripts, they will be classified as such. | No |
| Workshop | This includes course and exercise material, tutorials, workshops. | No |
| Docs | This includes notes, summaries, documentation, lists, presentations, ontologies, websites, guidelines, LaTeX files, lecture slides, project management (projects tab), publication PDFs, vignettes, books, and anything else falling into this category. | No |
| Template | Code, LaTeX or documentation templates/boilerplates that are not created during the research process according to our used definition. | No |
| Student work | Bachelor and master thesis related work, student coursework etc.; Actual research projects by students conducted under supervision (e.g. scientific internships) are not part of this category. Thesis work published on organizational accounts are counted as Research Software/Rscripts. | No |
| OtherRS | Research Software from researchers that were developed at other institutions. | No |
| Irrelevant | Test/dummy/Toy/Demo/course project/non-scientific code challenge repositories. Also includes other types of repositories, e.g. for storing artwork, data dumps of misc. projects, storage for configuration files, schemas, stylings, themes, moved or deleted repositories. | No |

# C Further user plots

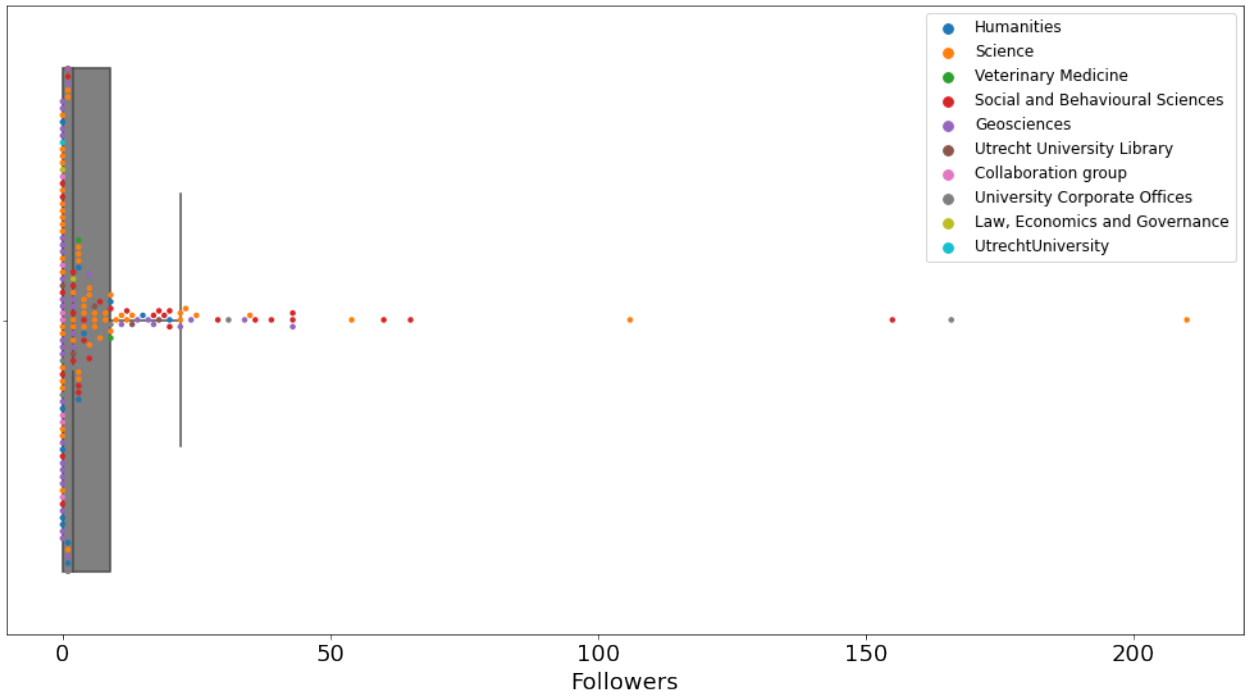


**Figure C.1:** Number of followers of users per faculty.

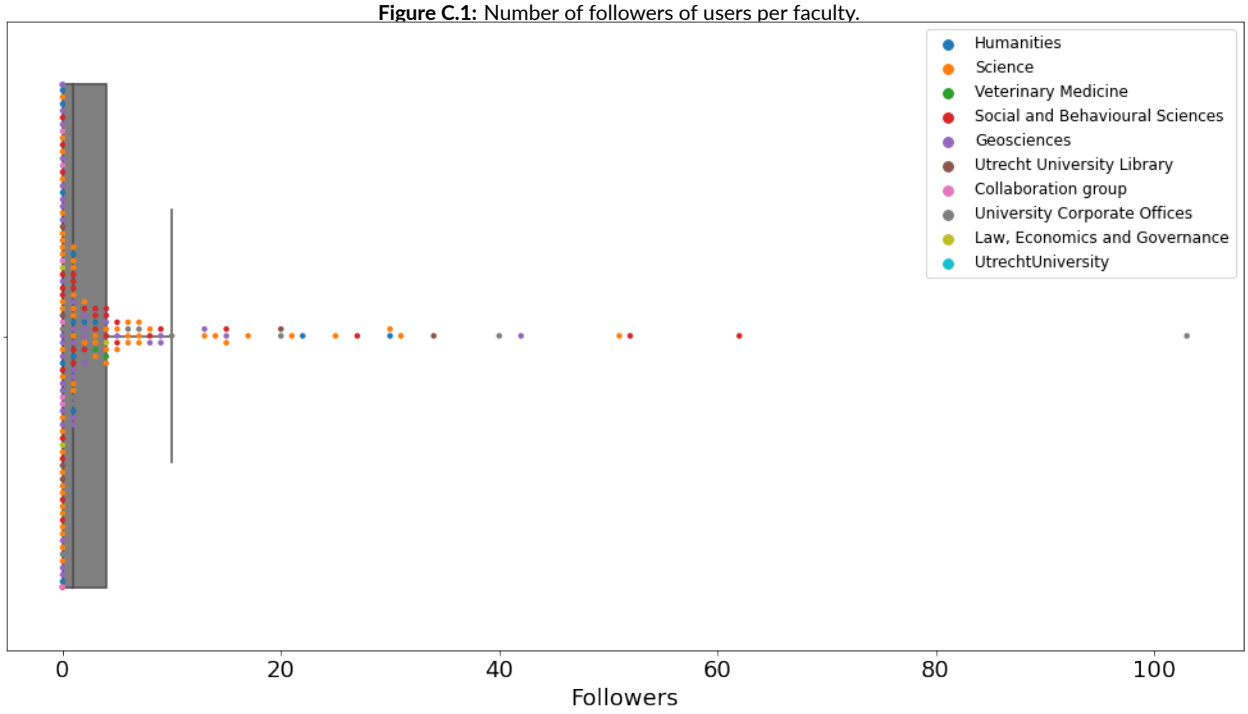


**Figure C.2:** Number of users a user is following per faculty.

# D Further repository plots

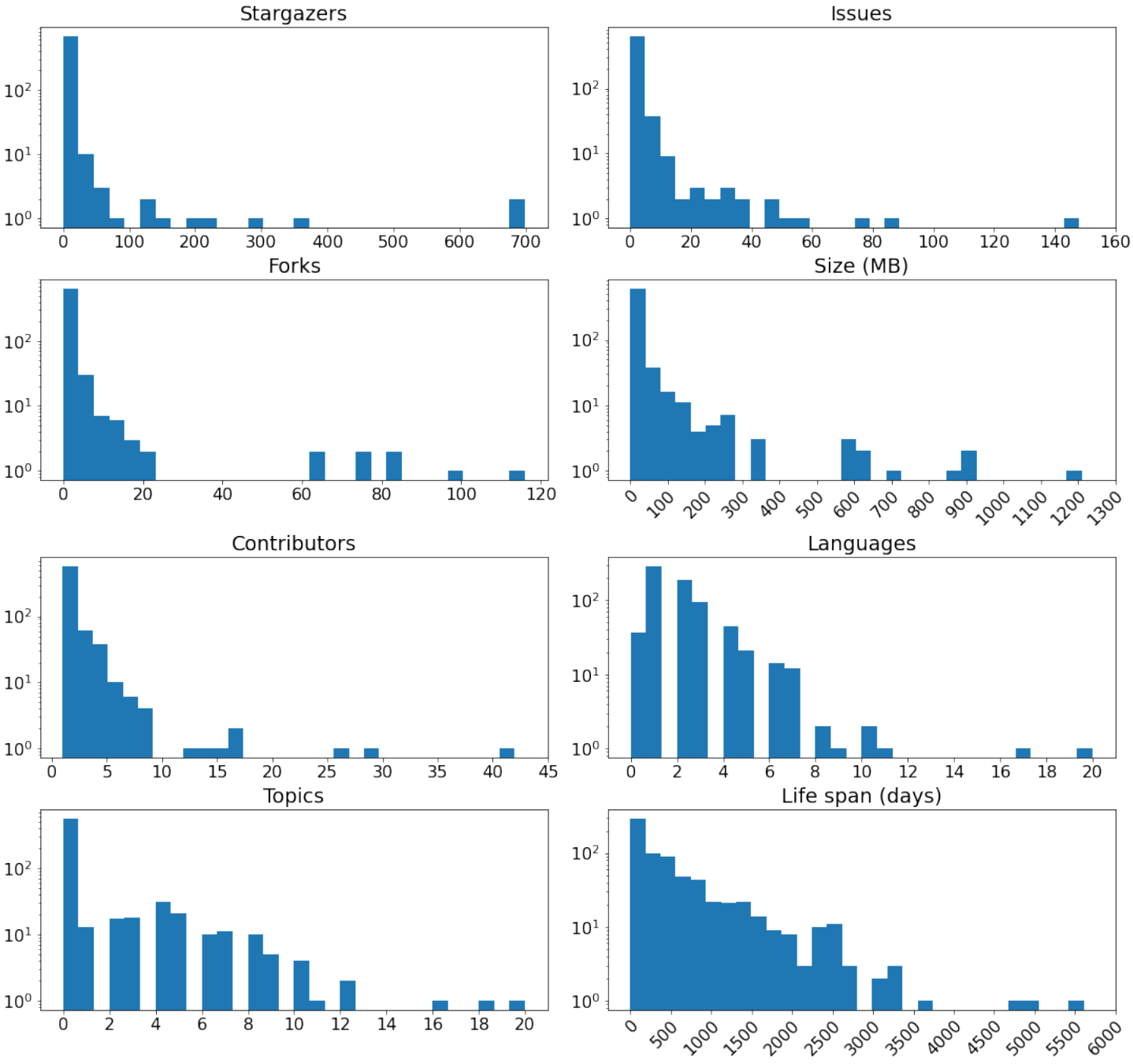


**Figure D.1:** Histograms for metrics that are considered research software. The y-axis is log-scaled.

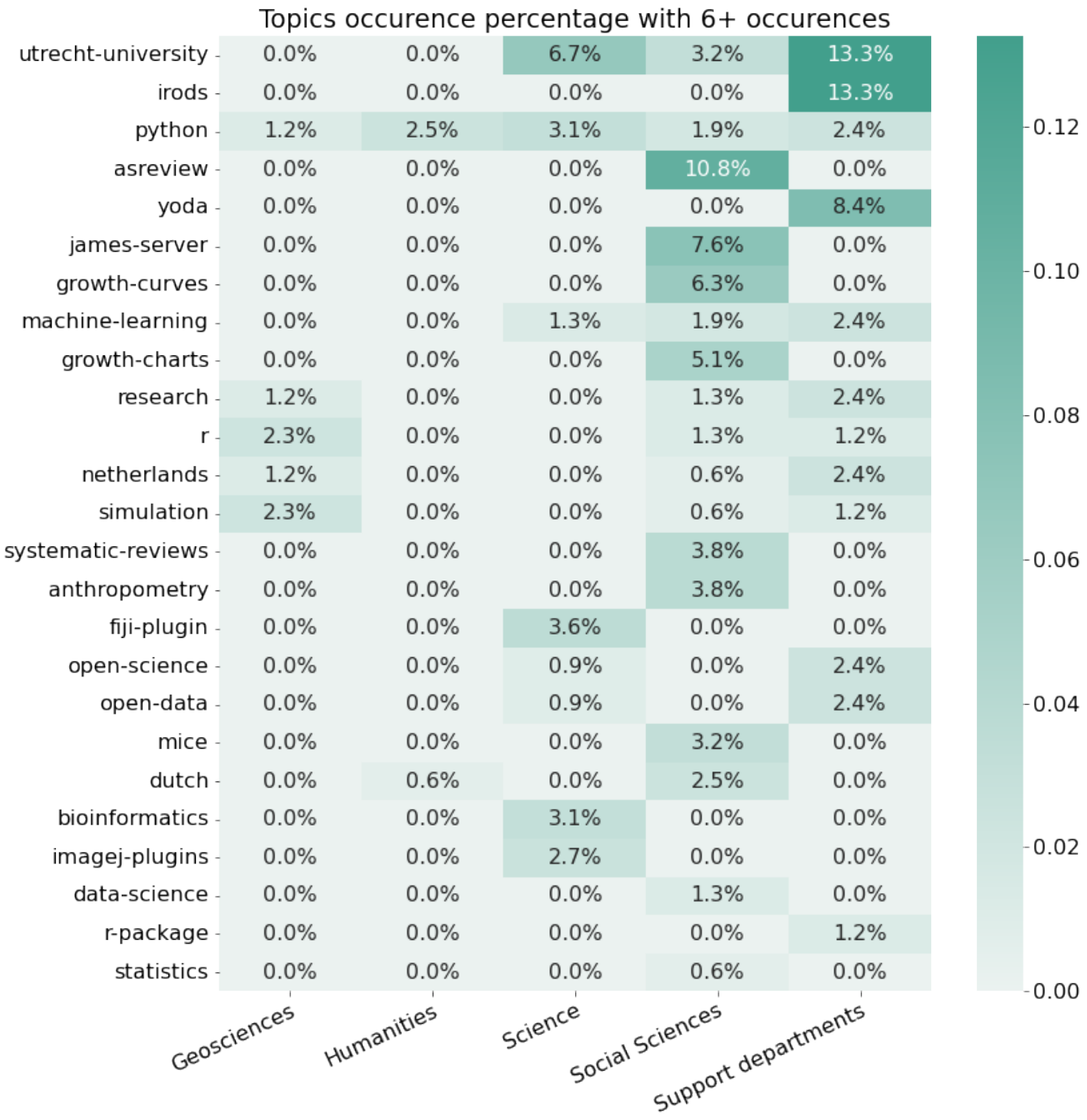


**Figure D.2:** Topics percentage for each faculty. Only topics that occurred more than 6 times were included.

# E Descriptive statistics per faculty

| | Stargazers | Issues | Forks | Size (MB) | Contributors | Languages | Topics | Life span (days) |
|---|---|---|---|---|---|---|---|---|
| Minimum | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 | 0.00 | 0.00 | 0.00 |
| 25th percentile | 0.00 | 0.00 | 0.00 | 0.02 | 1.00 | 1.00 | 0.00 | 12.00 |
| Mean | 2.84 | 3.02 | 1.49 | 16.96 | 1.22 | 2.19 | 0.62 | 473.46 |
| Median | 0.00 | 0.00 | 0.00 | 0.12 | 1.00 | 2.00 | 0.00 | 267.00 |
| 75th percentile | 1.00 | 0.00 | 0.75 | 5.98 | 1.00 | 3.00 | 0.00 | 662.00 |
| Maximum | 80.00 | 148.00 | 62.00 | 263.95 | 4.00 | 20.00 | 10.00 | 3623.00 |
| Skewness | 5.70 | 7.05 | 8.07 | 3.67 | 2.88 | 4.42 | 3.30 | 2.32 |
| Kurtosis | 33.76 | 51.33 | 69.78 | 14.09 | 8.77 | 26.72 | 10.97 | 5.91 |

**Table E.1:** Descriptive statistics of numeric variables for Geosciences

| | Stargazers | Issues | Forks | Size (MB) | Contributors | Languages | Topics | Life span (days) |
|---|---|---|---|---|---|---|---|---|
| Minimum | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 | 0.00 | 0.00 | 0.00 |
| 25th percentile | 0.00 | 0.00 | 0.00 | 0.08 | 1.00 | 1.00 | 0.00 | 39.25 |
| Mean | 0.58 | 2.92 | 0.18 | 9.52 | 1.79 | 2.36 | 0.25 | 591.39 |
| Median | 0.00 | 0.00 | 0.00 | 0.42 | 1.00 | 2.00 | 0.00 | 430.00 |
| 75th percentile | 0.00 | 2.00 | 0.00 | 4.92 | 2.00 | 3.00 | 0.00 | 816.50 |
| Maximum | 11.00 | 77.00 | 3.00 | 205.65 | 9.00 | 7.00 | 11.00 | 2796.00 |
| Skewness | 4.34 | 5.54 | 3.43 | 4.56 | 2.58 | 1.02 | 6.60 | 1.38 |
| Kurtosis | 21.09 | 35.84 | 12.59 | 28.83 | 7.79 | 0.45 | 48.36 | 1.23 |

**Table E.2:** Descriptive statistics of numeric variables for Humanities

| | Stargazers | Issues | Forks | Size (MB) | Contributors | Languages | Topics | Life span (days) |
|---|---|---|---|---|---|---|---|---|
| Minimum | 0.00 | 0.0 | 0.00 | 0.00 | 1.00 | 0.00 | 0.00 | 0.00 |
| 25th percentile | 0.00 | 0.0 | 0.00 | 0.11 | 1.00 | 1.00 | 0.00 | 38.00 |
| Mean | 8.71 | 1.3 | 2.42 | 38.85 | 2.08 | 2.17 | 1.17 | 590.09 |
| Median | 1.00 | 0.0 | 0.00 | 1.17 | 1.00 | 2.00 | 0.00 | 278.00 |
| 75th percentile | 3.00 | 0.0 | 1.00 | 11.75 | 2.00 | 3.00 | 0.00 | 764.00 |
| Maximum | 699.00 | 53.0 | 97.00 | 927.14 | 42.00 | 17.00 | 20.00 | 5609.00 |
| Skewness | 11.19 | 7.1 | 7.87 | 5.26 | 9.36 | 3.39 | 3.11 | 2.77 |
| Kurtosis | 141.37 | 56.8 | 66.58 | 32.71 | 110.80 | 17.78 | 12.14 | 9.64 |

**Table E.3:** Descriptive statistics of numeric variables for Science

| | Stargazers | Issues | Forks | Size (MB) | Contributors | Languages | Topics | Life span (days) |
|---|---|---|---|---|---|---|---|---|
| Minimum | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 | 0.00 | 0.00 | 0.00 |
| 25th percentile | 0.00 | 0.00 | 0.00 | 0.13 | 1.00 | 1.00 | 0.00 | 34.00 |
| Mean | 6.78 | 1.36 | 1.99 | 34.96 | 1.99 | 1.77 | 1.37 | 514.22 |
| Median | 0.00 | 0.00 | 0.00 | 1.45 | 1.00 | 1.00 | 0.00 | 336.50 |
| 75th percentile | 1.75 | 0.00 | 1.00 | 9.53 | 2.00 | 2.00 | 2.00 | 759.25 |
| Maximum | 350.00 | 36.00 | 84.00 | 870.33 | 29.00 | 7.00 | 18.00 | 3337.00 |
| Skewness | 8.30 | 4.91 | 7.93 | 5.09 | 6.92 | 1.75 | 2.66 | 1.86 |
| Kurtosis | 71.01 | 26.28 | 65.80 | 27.67 | 53.17 | 3.64 | 10.23 | 4.03 |

**Table E.4:** Descriptive statistics of numeric variables for Social and Behavioural Sciences

| | Stargazers | Issues | Forks | Size (MB) | Contributors | Languages | Topics | Life span (days) |
|---|---|---|---|---|---|---|---|---|
| Minimum | 0.00 | 0.00 | 0.00 | 0.00 | 1.00 | 0.00 | 0.00 | 0.00 |
| 25th percentile | 0.00 | 0.00 | 0.00 | 0.07 | 1.00 | 1.00 | 0.00 | 74.00 |
| Mean | 12.16 | 2.07 | 3.46 | 29.54 | 2.46 | 2.01 | 1.83 | 452.88 |
| Median | 0.00 | 0.00 | 0.00 | 0.56 | 2.00 | 1.00 | 0.00 | 256.00 |
| 75th percentile | 2.00 | 1.50 | 1.00 | 3.13 | 3.00 | 3.00 | 3.00 | 548.50 |
| Maximum | 694.00 | 49.00 | 116.00 | 1209.65 | 17.00 | 7.00 | 12.00 | 2375.00 |
| Skewness | 8.55 | 6.42 | 6.43 | 7.04 | 3.84 | 1.35 | 1.78 | 1.75 |
| Kurtosis | 75.39 | 49.14 | 43.42 | 52.86 | 16.16 | 1.89 | 3.08 | 2.48 |

**Table E.5:** Descriptive statistics of numeric variables for Support departments

This thesis was typeset using LaTeX, originally developed by Leslie Lamport and based on Donald Knuth's TeX. The body text is set in 11 point Egenolff-Berner Garamond, a revival of Claude Garamont's humanist typeface. A template that can be used to format a PhD dissertation with this look & feel has been released under the permissive AGPL license, and can be found online at github.com/suchow/Dissertate or from its lead author, Jordan Suchow, at suchow@post.harvard.edu.