–B.Sc. Thesis–

Creating a Pipeline for Reproducible Evaluation Report Generation

Freie Universität Berlin

Institute of Computer Science

Human-Centered Computing (HCC) Research Group

Alexander Rudolph

Date of Submission: 17.05.2024

Supervisor:

Prof. Dr. Claudia Müller-Birn[*], Freie Universität Berlin, Germany

Examiner:

Prof. Dr. Claudia Müller-Birn[†], Freie Universität Berlin, Germany

Barry Linnert [‡], Freie Universität Berlin, Germany

[*]Department of Mathematics and Computer Science, Human-Centered Computing Research Group

[†]Department of Mathematics and Computer Science, Human-Centered Computing Research Group

[‡]Department of Mathematics and Computer Science, Software Engineering

[§]https://creativecommons.org/licenses/by-nc-sa/4.0/

Abstract

To ensure lecturers can improve their teaching quality, the Faculty of Mathematics and Computer Science employs an evaluation process every semester, during which students can evaluate their lecturers anonymous and directly. In addition to individual results, the faculty also manually creates an evaluation report for all institutes belonging to it. To support the evaluation committee in generating the report, a reproducible pipeline will be created to automate this task as much as possible.

Special focus will be applied to the concepts of code understandability and (computational) reproducibility, while following modern design, documentation, coding guidelines and specifications. This ensures the pipeline is usable by anyone and continues to stay so in the future.

Along with the pipeline, a prototype for a dashboard will be developed with minimal functionality to demonstrate how the processed data can be used to convey results in a way that extends the current evaluation report.

Contents

1. Introduction

Evaluations in the academic context provide important feedback to lecturers that can help them better assess student needs as they continue teaching their subjects. Providing anonymous feedback will not only give students a useful tool to express their likings and grievances, but will also enable lecturers to include the results in their teaching process. Understanding evaluation results not just as a good or bad grade for themselves, but rather as the base for an open discussion with students, helps raise student satisfaction by giving them the chance to influence the curriculum [5].

While individual lecturers only need their personal results, the faculty itself needs an overview of all courses to determine if teaching quality improves or decreases over time. Using evaluation reports can provide a comprehensive overview of all courses without having to execute the time-consuming task of analyzing every individual evaluation result themselves. To improve decision-making this visualization should encode as much information as possible, while still staying comprehensive to enable the decision-making process to move fast, but also efficient.

1.1. Motivation

Currently, most steps in the generation of an evaluation report at the faculty contain manual labor, where a single person uses `Excel` to interpret and clean the data. This process not only takes a long time, since it's repeated multiple times to validate the results, but also isn't transparent. Since the process doesn't follow a fixed scheme and isn't documented anywhere, it is impossible to reproduce, especially if the person conducting the process stops working for the institute. Additionally, this way of generating the report complicates the possibility to put results into historical context, due to the amount of additional workload that would be introduced. Creating a pipeline to handle the data cleanup and modification allows the process to become more transparent, since the steps that produced a result can be retraced by following the code steps. Reproducibility additionally enables the current and future personnel to understand the process and to keep it uniform, while also minimizing their workload with the report generation itself.

1.2. Goal

During this thesis, a pipeline will be implemented, which employs the concept of reproducibility to enable current and future members of the evaluation committee to get a historical development for evaluation performance in their faculty. To achieve this goal, the concept of reproducibility will be introduced (see subsection 2.3) and applied during the development process. In order to elicit requirements for the new pipeline, interviews will be conducted with key personnel from the institutes.

2. Theoretical Background

2.1. The Evaluation Process

According to the supplementary text provided with evaluation reports [24], evaluations are conducted in the second third of every semester. Lecturers are provided with a QR code they can distribute in their courses, which leads students to the evaluation page. Based on the course type, students fill out one of four different types of questionnaires. For lectures, tutorials and seminars they contain mostly similar questions, but also questions tailored specifically to the course type. In evaluations for lectures students are asked about the lecture contents and its difficulty, while the tutorial questions rather ask about the quality of interaction between the lecturer and the students. Seminar questionnaires are structured similarly to the ones provided for lectures, with the addition of asking about the topic itself, since seminars typically focus on a single topic, whereas lectures contain multiple.

If any lecturer is new at the faculty, they won't get any of the three types mentioned above, but rather get distributed a questionnaire assessing teaching competence of the new lecturer. This questionnaire contains all questions that can be found on the other three types and applies special focus to didactic capabilities of the lecturer.

Typically, students have about two weeks to fill out these forms before the links aren't accessible anymore. After the deadline the current approach begins, where the data gets transferred into the evaluation software, the individual results of lecturers get distributed back to them and the evaluation report gets generated.

2.2. The Current Approach

The requirement elicitation interviews conducted in subsection 3.2.1 reveal that the current process of creating the evaluation report contains almost exclusively manual labor. The Advisor for Studies and Teaching explained how they use the internal software `Zensus` [4] to extract the evaluation data into Comma Separated Values (CSV) files, which are used to manually create a minimal evaluation report.

This report follows a fixed scheme, where the first part contains text that is similar every semester. It explains how and when the evaluation was conducted, how many people started teaching this semester and finally statements about the participation of students.

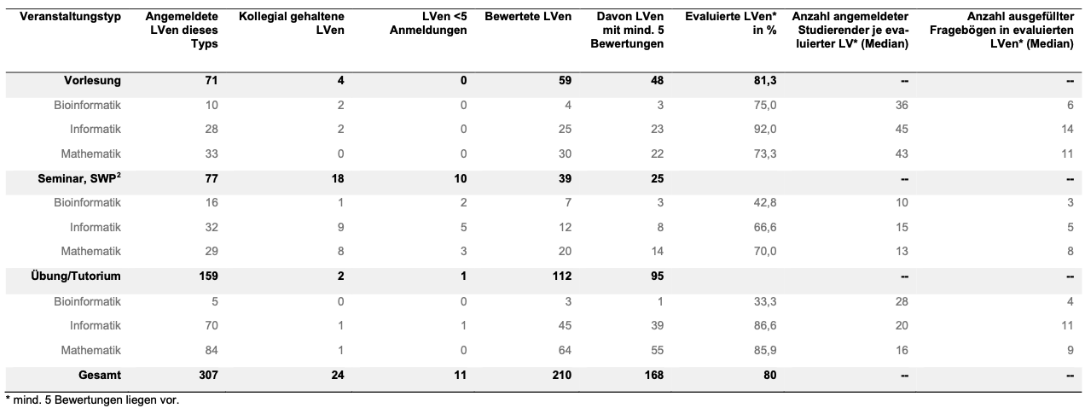
Following this section, a table is presented, which contains specific values about participation and provides most information about courses (cf. Figure 1). The table follows the same structure every semester, where the rows relate to the courses split into the three types of **lecture**, **seminar** and **tutorial** and then splits these types into subcategories that represent the three institutes of **bioinformatics**, **computer science** and **mathematics**.

Table columns represent the numeric values for these rows. In the first three columns informational values about the courses themselves are displayed. They denote how many instances of the specific types of courses were registered for the corresponding institute, how many of these were held cooperative by at least two lecturers and how many courses had less than five sign-ups.

The next three columns give information about the participation of students in the evaluation. Values included here, display how many courses were evaluated for a type, how many of the evaluated courses had at least five student responses to the evaluation and the percentage of how many courses were evaluated overall.

In the final two columns medians for the number of students registered for evaluated courses and the number of returned evaluations per course are provided.

The evaluation concept states that, due to data privacy of the students, courses only get their evaluation responses, when at least five students took part in the evaluation [17]. Courses that don't get an evaluation response aren't included in the report, since they wouldn't contribute to the overall statistic. Thus, some columns mentioned above contain the "at least five responses" constraint, where applicable.



Veranstaltungstyp	Angemeldete LVen dieses Typs	Kollegial gehaltene LVen	LVen <5 Anmeldungen	Bewertete LVen	Davon LVen mit mind. 5 Bewertungen	Evaluierte LVen* in %	Anzahl angemeldeter Studierender je evaluierter LV* (Median)	Anzahl ausgefüllter Fragebögen in evaluierten LVen* (Median)
Vorlesung	**71**	**4**	**0**	**59**	**48**	**81,3**	**--**	**--**
Bioinformatik	10	2	0	4	3	75,0	36	6
Informatik	28	2	0	25	23	92,0	45	14
Mathematik	33	0	0	30	22	73,3	43	11
Seminar, SWP^2	**77**	**18**	**10**	**39**	**25**		**--**	**--**
Bioinformatik	16	1	2	7	3	42,8	10	3
Informatik	32	9	5	12	8	66,6	15	5
Mathematik	29	8	3	20	14	70,0	13	8
Übung/Tutorium	**159**	**2**	**1**	**112**	**95**		**--**	**--**
Bioinformatik	5	0	0	3	1	33,3	28	4
Informatik	70	1	1	45	39	86,6	20	11
Mathematik	84	1	0	64	55	85,9	16	9
Gesamt	**307**	**24**	**11**	**210**	**168**	**80**	**--**	**--**

* mind. 5 Bewertungen liegen vor.

Figure 1: Overview table in the evaluation report from the summer term in 2023 [24]
Columns marked with a star (*) denote that they had the "at least five student responses" constraint.

In the interview with the Dean of Study, they additionally revealed that the deanery evaluation that should happen every semester, as mentioned by the evaluation guidelines of the institute [17], is currently not conducted. The presented example of such an evaluation in the guideline has never reached the dean. However, they could not say if their predecessors received such documents. They did note that they see the overall evaluation report when they participate in faculty council meetings, in which the evaluations report is presented.

2.3. Introducing Reproducibility

The main focus of this work is to deliver a result that is reproducible. Designing and implementing the pipeline this way, allows other people to verify results and correctness. Before creating a reproducible software, it is important to define what *reproducibility* is.

This thesis aims to use definitions according to Kitzes et al. [10]. In their book they conducted 31 case studies, which employed a reproducible research workflow. They introduce the concept of *computational reproducibility*, defined as "A research project is computationally reproducible if a second investigator (including you in the future) can recreate the final reported result of the project, including key quantitative finding, tables, and figures, given only a set of files and written instructions" [10]. This definition later gets more generalized to the essence, that research is reproducible, if other researchers can use the original raw data and methodology to create the same workflow and thus reaching the same result. The aim of this concept is to create a workflow, which allows the analysis of new data with the same workflow. Since this thesis attempts to create a pipeline that works with past, present and future evaluation data, both concepts will be applied. To enable other users to understand and use the pipeline, it will be created *reproducible*, by providing the necessary code that ensures evaluation reports can be generated in the future, provided the questionnaires or data structure won't change significantly. This is in line with an answer provided by the Evaluation Officer, who stated that ideally the pipeline will be self-explanatory, so they could give the code to any external person, who then could comprehend how the results are created.

This concept of reproducibility should not be confused with the term of *replicability.* According to Kitzes et al. [10] any software is replicable, if the provided code produces the same output, regardless of the person executing it. Reproducibility wants to recreate results by using the same methods, while replicability recreates results by using the exact same steps.

To ensure the workflow can be reproducible, the key points from Kitzes et al. [10] will be applied. They suggest using automation to the degree that the person accessing the workflow only needs to press one button or spend an equivalent of minimal effort to produce a result, without creating a monolith. To achieve this, the steps can be split into multiple files or multiple cells in `Jupyter Notebooks`. However, this introduces the need to highlight the starting point of the whole workflow. Employing any type of wiki or `READMEs` helps people to access the workflow for the first time. Supplying *Makefiles* for example, can also simplify interaction with the workflow, by providing commands that execute the stept. Another key focus of reproducible workflows is availability. Using a publicly accessible version control system like `GitHub`[1] or `GitLab`[2] not only provides interested people with the source code, but also enables utilization of the git tag system, where current and past versions and their differences can easily be labeled. When using

[1]https://github.com
[2]https://gitlab.com

a public version control system, a corresponding license should be supplied to ensure derivatives of the original workflow stay publicly accessible. By choosing a Copyleft license, anyone using the workflow in their own software in any way, has to make their source code available under the same terms as the original workflow. This helps with preserving any future iterations of the workflow, by forcing the source code to stay open to the public [11].

The resulting pipeline will follow these key points. It will be hosted publicly on the `GitLab` instance of the faculty (see Appendix A) with the copyleft *GNU GPL v3 License*[3] forcing the source code to stay open. Additionally, the `Wiki` feature in `GitLab` is used to provide documentation for the workflow. To provide in-code documentation, the option to create markdown cells in `Jupyter Notebooks` will be employed. For simplified access with the pipeline, a *Makefile* is provided that combines installation, building and execution into the command `make install` and `make run_pipeline`.

[3]https://www.gnu.org/licenses/gpl-3.0.en.html

3. Analysis

3.1. Analyzing the Current Approach

In the interviews, the Advisor for Studies and Teaching revealed how they create the evaluation report manually, and according to them, this method contains a few caveats.

They mentioned that the `Zensus` software they use can't automatically find and mark lectures that were held cooperative, so the advisor has to search for and count them by hand. The resulting CSV file from `Zensus` contains all responses, which means they have to manually group courses belonging together, filter results and count responses.

This process, while trivial, can be error-prone due to the amount of data in these files. For example, in the summer term 2023, the three files for computer science, bioinformatics and mathematics contained 1028, 92 and 1196 lines (= number of individual responses) respectively, adding up to around 2316 data points per semester. Using these data points, the advisor then has to filter out courses that received less than five responses, which means counting rows belonging to the same course by hand. The remaining responses then need to be grouped into the respective courses, to be able to get the number of responses for every course to use in calculations. Finally, to ensure the results are accurate, the process needs to be repeated multiple times, to guarantee errors weren't made. The advisor noted in the interviews that this process can take them up to three hours.

Executing these steps manually invites errors. Comparing the data with the results on the evaluation report shows that in some semesters the median includes all registered courses, while in other semesters it's calculated using courses with at least five responses. Similar problems arise when looking at the column that reports how many courses were evaluated, where sometimes all registered courses are used as base value, while courses that had at least five registered students are used in other reports. Since these mistakes aren't obvious to the naked eye, it becomes near impossible to notice them when calculating results manually.

3.2. User Requirements

In order to create the resulting software in a manner that is user centered, a design process was employed that aims to create software in cooperation with the stakeholders. By following the key principles for a User-Centred System Design (UCSD) process proposed by Gulliksen et al. [7], the development cycle will be guided by the user and their goals, ensuring the result is to their satisfaction. USCD notes that not only should the user requirements be extracted early in the development, but the user should always be involved. Based on this technique a process proposed by International Organization for Standardization (ISO) for human-centred design is applied (cf. Figure 2).

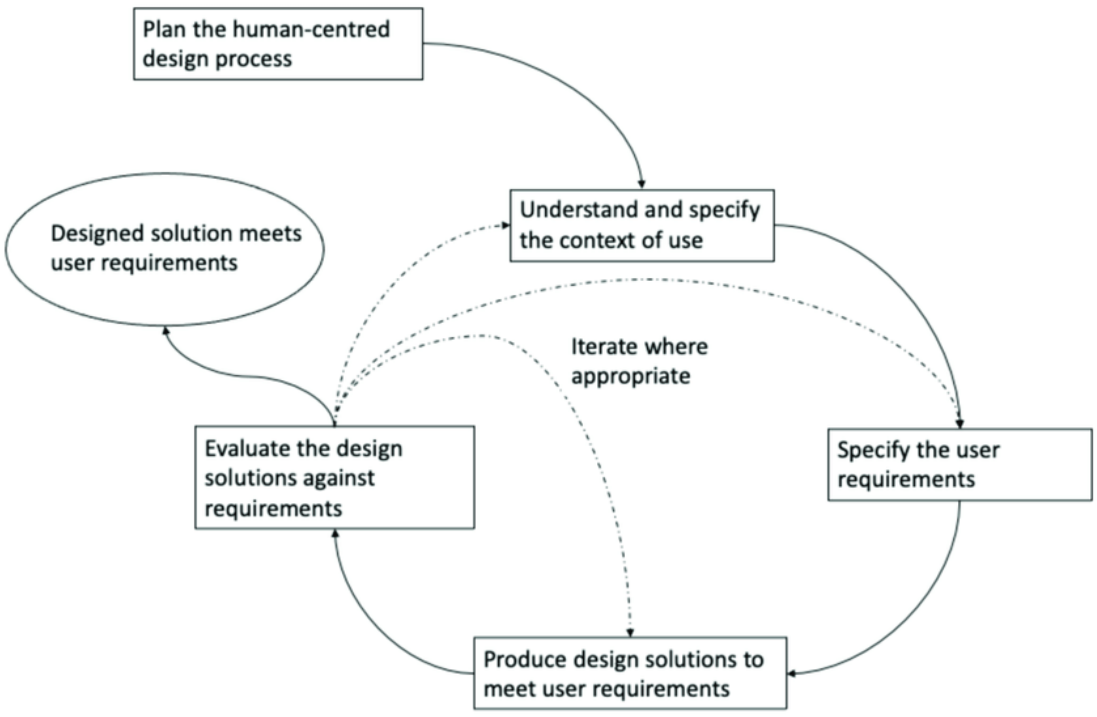

Plan the human-centred design process
Understand and specify the context of use
Designed solution meets user requirements
Iterate where appropriate
Evaluate the design solutions against requirements
Specify the user requirements
Produce design solutions to meet user requirements

Figure 2: Human-centred design process based on ISO 9241-210 Ergonomics of human-system interaction-Part 210: Human-centred design for interactive systems [8]. The solid lines represent transitions that must occur and the dotted lines are transitions that may occur depending on how the processes evolve.

To understand the context of evaluation reports and to identify the needs and wants from people working with the pipeline or the generated results, requirement elicitation interviews were conducted with key personnel at the institutes. Applying concepts from Scheinholtz et al. [18], the interviews provided only a few questions that mainly are intended to act as a transition between topics, while the actual interview relied on reactions and replies by the interviewer to what the interviewee said. This leads the interview to be structured more like a conversation between both participants, thus causing the interviewee to expose information that they thought was trivial or didn't need explaining.

Different tags were applied to questions as classification, guaranteeing they can be asked in the right context. In the same publication Scheinholtz et al [18] suggested that questions can be split into three groups which classify the question type (cf. Figure 3). The first group ("Open vs. Closed") identifies the questions' specificity. Open questions allow the person answering them a lot of room on the topic, while closed questions restrict the answer possibilities. Open and Closed Questions can be tagged even more precisely by using the adjectives highly and moderately to specify the degree of openness or closeness. The second group ("Neutral vs. Leading") mark the interviewers intentions.

When asking leading questions, the interviewer may already have an answer in mind they want to hear, while neutral questions are used as ”normal” interview questions that need an answer. Finally, the last group (”Primary vs. Probing”) exclaims the current position of the interview. Asking a primary question opens up a new subject different to the one discussed before, and asking a probing question attempts to extract more answers out of the interviewed person [18]. Most questions were labeled as primary, since probing questions rather arise during the interview itself and can only be identified after it concluded. Closed questions are mainly used for extracting (non-) functional requirements, while open questions are employed when the interviewed person should talk about their relation to the evaluation process.

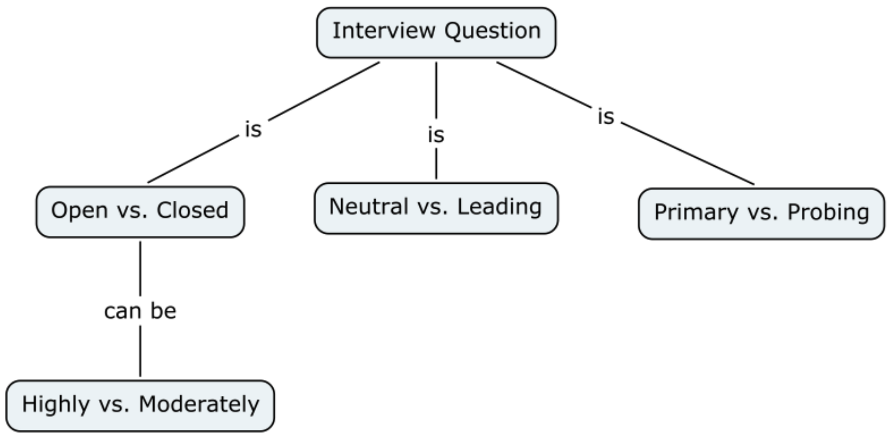


Figure 3: Question Qualifications according to Scheinholtz et al. [18]

3.2.1. Interview 1 - Initial Requirement Elicitation

Overall six members of the institutes have been identified as suitable candidates for requirement elicitation interviews due to their direct involvement in the evaluation process or because of their experience with evaluations at the university. The three key interviewees consisted of the Evaluation Officer, who oversees the general procedure of evaluations at the institute, the Advisor for Studies and Teaching, who conducts the evaluations and creates the resulting report, and the Dean of Study, who is tasked with improving quality of teaching. Additionally, three lecturers of the three institutes were chosen, who once were in the position of Dean (of Study).

All participants received the same base set of questions that asked about the current evaluation report and how they think it can be improved. To ensure everybody could provide ideas they normally may not disclose, the *headstand method* was employed, where the question of ”How would an ideal evaluation report look like to you” was negated into

”How would an evaluation report look like to you, so you couldn’t make any statements about teaching quality” to force interviewees out of their perspective [9].

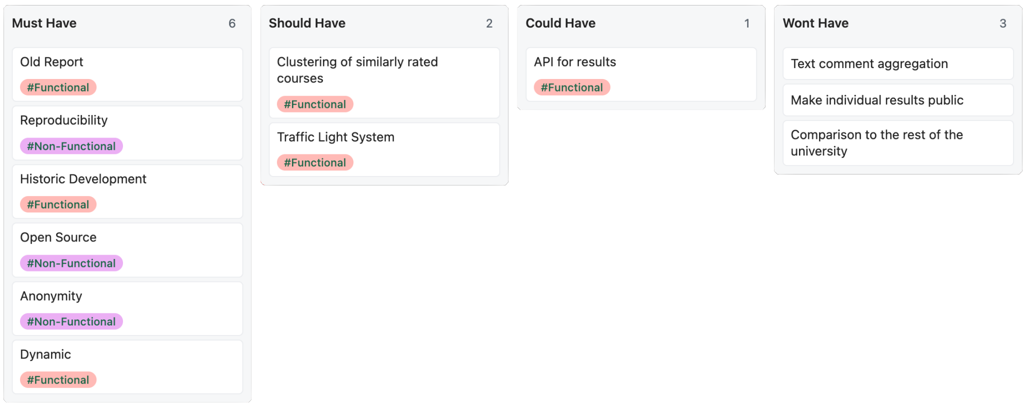
In addition to the base set of questions, each type of interviewee got questions specifically tailored to their job position. The Dean of Study was asked about how they use the results of the evaluation report as a tool to fulfill their task of improving teaching and studying. Due to their involvement in the process itself, the Evaluation Officer and the Advisor for Studies and Teaching were asked about how they partake in the evaluation process and what steps they take from the beginning of the evaluation up to the final evaluation report. The lecturers of the institute were asked how they use evaluations overall in their work and if they pay attention to the results of the evaluation report.

During these interviews all interviewees confirmed that the deanery evaluation was never conducted. Where applicable, the interviewee was asked questions about their preferred version of such an evaluation. Based on the answers, the deanery evaluation would contain the same information as the evaluation report itself, with the addition of a ranking of all courses. This ranking would then be used to award good teaching and to identify teaching deemed insufficient by students. In opposition to the public evaluation report, the deanery evaluation is only shown to the dean and can contain personal data, thus enabling the ranking.

Once the interviews were concluded, requirements got extracted and categorized by using the *MoSCoW* prioritization. According to the Agile Business Consortium [2], *MoSCoW* stands for **M**ust Have, **S**hould Have, **C**ould Have and **W**ont Have. Each category has a ruleset, which determines what type of requirements are supposed to be in it. Requirements that are in the Minimum Usable SubseT (MUST Have) denote core features that ensure usability at the point of delivery. Without these features the software could either be illegal, dangerous or simply just not viable enough. Once the Must Haves are implemented, the software can be expanded with Should Haves. These requirements aren’t necessary, but would improve user experience noticeably. After Must and Should Haves are implemented, the software can be improved by introducing the Could Haves. They will improve the software similarly to Should Haves, but will be less noticeable if implemented or left out. Users could live without them, but also wouldn’t mind haven these minor features. Finally, the category of Wont Haves are requirements that will not be implemented in either this release cycle or ever. This category may also simply hold features that will be addressed, when deemed possible. Keeping this category can help users and developers alike by acting as a list that can be looked at to prevent duplicated requirements.

The extracted items were put into the categories as seen in Figure 4 and also tagged as a functional or non-functional requirement, where functional requirements are features that specifically need to be implemented during development, while non-functional requirements describe attributes of the resulting pipeline [22].

As visible, the MUST is the old report, but with the enhancement of displaying *historic development*. Non-functional requirements like *Open Source* or *Anonymity* can easily


Must Have 6

Old Report #Functional

Reproducibility #Non-Functional

Historic Development #Functional

Open Source #Non-Functional

Anonymity #Non-Functional

Dynamic #Functional

Should Have 2

Clustering of similarly rated courses #Functional

Traffic Light System #Functional

Could Have 1

API for results #Functional

Wont Have 3

Text comment aggregation

Make individual results public

Comparison to the rest of the university

Figure 4: MoSCoW prioritization of requirements

be realized by hosting the source code publicly, while withholding the source data and removing lecturer names when results are displayed anywhere. *Reproducibility* is guaranteed by following the concepts mentioned in subsection 2.3, which in turn helps with keeping the pipeline dynamic, by providing code that can easily be understood and thus expanded. Ensuring these key features will not only speed up the generation of reports, but also enhance them by providing historic context.

Requirements that are possible and enhance the visualization of the report are put into Should Have. They were wished by a few people and would improve the results noticeably. Providing the results of the report generation as an *application programming interface (API)*, where users can simply request results by using the provided endpoints, would enable people not involved in the evaluation process to interact with results to create own visualizations over a standardized endpoint. Additionally, this would apply an *Open Science* approach to the evaluation where the open code of the pipeline in combination with the results make the research process available to everyone just like the UNESCO suggest in *Understanding open science* [21].

Finally, requirements put into Wont Have will be discussed further in subsection 5.2.

3.2.2. Interview 2 - Prototype Presentation

The second interview served the purpose to produce a design that satisfies the users and evaluate this design with them (following the steps in Figure 2). For this, mockups were created, that implemented the requirements extracted in the first interviews and presented to the same interviewees (see Appendix B). To assess the validity of the designs, the interviews were designed with the *Think Aloud* method, where participants will normally be presented with software unknown to them and tasked with navigating through it while vocalizing their thoughts during the experience [12]. In this thesis however, the interviewees are presented with the mockups and asked to describe what information they can get from it. Analogue to the original intention of the method, this interpretation will show if the designs are intuitive, convey the right data points and lead interpreters to the intended conclusions.

For this interview, every aspect of the *MoSCoW* prioritization, except the Wont Haves were included. Must Haves like *Open Source* and *Reproducibility* can't be illustrated and thus won't be visible in the mockups.

Historic development was shown in multiple examples. It was visualized with a line chart, bar chart and visible in the box- and violin plots. Most of the interviewees confirmed, that a line chart is a good way to display historic data, while the bar chart obfuscates this goal. The traffic light system was employed in a table where arrows display historic development. To convey if the development is positive, negative or neutral, the arrows were colored green, red or yellow respectively. They would receive their color based on intervals in which the development resides, which is the same for every course type. Interviewees noted that using a system like this can obfuscate interpretation. If a course was colored red for a long time and now starts to ascend into the yellow category they still seem to perform badly, even though they managed to improve. In order to prevent confusion, this should be mentioned when the table is used on an evaluation report.

The "Should Have" of clustering courses was done by a scatter plot. In order to enable comparison of values and to minimize the amount of data points, questionnaires were placed along dimensions, rather than individual responses to questions. These dimensions are derived from the lecture competence questionnaire, which splits its 44 questions into three dimensions that group questions into "Conveying Knowledge and Supporting Understanding" (Dimension A), "Motivate and Create an Atmosphere conductive to Learning" (Dimension B) and "Control of Interaction in the Learning Group" (Dimension C). Since every question on the teaching competence questionnaire can be mapped onto the other three, these categories can be applied universally. In order to create two similar values that can be compared along two axes, the initial three dimensions have been split into two subdimensions each. Since the lecture competence questionnaire not only splits questions into dimensions, but also categorizes similar questions, these categories can be assigned to subdimensions. This resulted in the six dimensions of "Course Design" (A.1), "Student interaction" (A.2), "Catering to Students" (B.1), "Motivating Students" (B.2), "Rules of Conduct" (C.1) and "Time management" (C.2). What questions belong to what category can be found under `src/db/questions.json` in the `GitLab` repository (see Appendix A). With these categories, the mockup places courses based on their score in the dimensions onto the coordinate system and clusters them with similarly placed ones. Almost all interviewees stated that they like the idea, but would rather like to see a scatter plot, where each lecture is placed by its individual score, without creating clusters. This not only allows to get a more detailed overview about how courses are doing, but will also create clusters on its own, if courses are close enough to each other.

Finally, box- and violin plots also took advantage of the dimensions mentioned above and visualized the comparison between institutes over time. The only feedback received on this type of visualization is that both box and violin plots are hard to read for people not familiar with them, thus a legend should be provided.

Overall the second interview concluded that the mockups are useful and the dimension system is viable. However, most interviewees mentioned that the design choice of coloring the institutes gray for computer science, green for bioinformatics and blue for mathematics, provided a visual challenge and should be more accessible in future visualization to include people with color blindness.

By omitting course and lecturer names on visualizations, anonymity is guaranteed, since courses are displayed as a group instead of individuals. However, this design choice got negative responses from almost all interviewees, claiming that evaluation results should be publicly accessible for all members in the evaluated institutes. While this thesis will keep names of lecturers and courses hidden as part of the requirements by both the university and members of the evaluation committee, the cleaned data will still be kept in a way that keeps the course and lecturer names intact, providing the necessary flexibility, in case this requirement should change in the future.

4. Implementation

4.1. Evaluation Data

After evaluations are finished, the data of the responses is exported from `Zensus` into CSV files. The output is split into three separate folders representing the institutes, where each folder has subfolders for evaluated semesters. Every generated folder contains four files, *daten.csv*, *kommentare.csv*, *mittelwerte.csv* and *summen.csv* with the last three files representing student comments, averages and sums respectively. *daten.csv* is a file containing all data, which can be used to generate the other three files. Since the first file contains all data, the pipeline only uses that file and transforms it as needed. An example of this file can be found in the repository (see Appendix A) under `raw_data/data.example.csv`.

The data in the file contains about 152 columns that follow a specific schema. In the first columns the provided data contains metadata of the course. Here the course name, number, type, lecturer and additional data that is used by `Zensus` is displayed. The remaining columns in the file contain all questions from the four questionnaires, with each column corresponding to one question. In every file duplicated questions can exist, since every question in every questionnaire is listed sequentially, thus containing duplicates if questions exist on two or more questionnaires. An example dataset with all columns and dummy data can be found in the *raw_data* folder in the GitLab Repository (see Appendix A). It should be noted that the CSV files generated by `Zensus` have semicolons as separators and are encoded in ISO-8859-1 and need to be opened with these constraints in mind in order to keep special characters like umlauts and to maintain the column structure.

Additionally, an extra file is needed per semester, in which the number of students and lecturers for each course is denoted. The Advisor for Studies and Teaching mentioned that the format of these files evolved over time, when they encountered complications with the schema they used before. This required to normalize these files, as every semester contained a different way to store the data. Every file was normalized by hand, since the schema was impossible to normalize with code. To clean up the files most column names were uniformed, since most of them changed over the years. The course and questionnaire type sometimes contain the key of "LeKo" in both columns. This complicates the identification of the course by its type, as it becomes impossible to infer if it is a lecturer, seminar or tutorial. The course type would then be changed to *Vorlesungen* (lectures), *Seminare* (seminars) or *Übungen* (tutorials) according to their entries in the course directory of the university, to enable the categorization later in the pipeline. Normalizing was oriented on the layout of the most recent semester schema, since it is the one currently used. The final schema contains the course number, course name, questionnaire type, course type, lecturer, number of participants, number of lecturers and the institute name. An example can be found in the `raw_data/participants/example.csv` file in the source code (see Appendix A).

4.2. Technology

As mentioned in subsection 2.3 the chosen file type is `Jupyter Notebook`, which enables to write the application code using `python` that can be separated into multiple code cells to create visual groups of steps that belong together. Additionally, the cells can be used as Markdown fields allowing to expand the code with textual explanations for sections. While the Notebooks were employed where possible, a few files remained normal python files, since they provide functionality not possible with `Jupyter Notebooks`, like providing constant variables or functions that interact with the API (see subsection 4.5) that are used by multiple files.

In order to simplify interaction with data, the popular `python` data analysis tool `pandas` was used. `Pandas` can convert data into a *DataFrame*, which essentially is a table in which the residing data can be filtered, manipulated and modified without having to change the original CSV file [23].

4.3. Data Cleanup

The raw data contains a few obstacles. Since the file structure changed over time, the data isn't uniform across semesters. For example the column denoting the questionnaire type has four possible entries in the most recent data. Here "VL" is used for lectures, "S" for seminars, "Ü" for tutorials and finally "LeKo" for teaching competence. However, past data does not follow this rule. In these files, lectures are also sometimes identified by "V", "Vl" or "VL+Ü". This creates three possible values to be expected in the data, that need to be handled. To prevent the pipeline itself having to encode these types and additionally to provide support for future notation changes, the cleanup script applies a function to the column that uses a dictionary to normalize names. This dictionary provides the possibility to change future notation by simply expanding or changing it. Currently, this step normalizes the names for lectures mentioned before to "VL", but also changes special combinations like "VLeko" and "SLeko" to "VL" and "S" respectively. When looking at the course category, the courses in the most recent data file are denoted in their plural form ("Vorlesungen", "Übungen", "Seminare"). However, in past data some courses are singular. The pipeline takes a similar approach to the questionnaire type and maps a dictionary to the singular nouns, converting them to plural.

To prevent the pipeline having to deal with large files, the cleanup script removes columns deemed unnecessary. These are indexed in an array that can be expanded or shrunk as needed.

During development, it became apparent that some course numbers have leading or trailing whitespaces. These can hinder the pipelines' ability to identify courses, as the whitespaces will fail equality checks with the same course number without whitespaces. Thus the cleanup script removes all leading and trailing whitespaces in the lecture number column. A similar problem exists with question columns, where some questions contain trailing whitespaces, which becomes a problem when attempting to merge questions, since duplicates could be overlooked. The cleanup script removes these whitespaces the same way as it does with the course numbers.

The next cleanup step merges questions. Due to the fact that the columns in the data represent the questionnaires in order, some questions that are on multiple questionnaires, will become duplicated columns. This does not only cause `pandas` to append numbers to the duplicates, but also storing the data this way leaves large gaps in the values. Courses that were evaluated by one type of questionnaire, won't be evaluated by another one. This means lectures for example have data in 27 questions columns and no data in the remaining 113 columns allotted for questions. Since data is organized this way, duplicated questions can simply be merged into one column, without the danger of overwriting values. This reduces the file size and ensures that column names are truly unique. After the merge took place the duplicated columns are dropped. Once all columns have been merged, they get replaced with `Universally Unique Identifiers` (UUIDs) to enable the pipeline to interact with the API defined in subsection 4.5.

This data then is split into the questionnaire types and saved in the `clean_data` folder along with a file containing all the data. The original structure of separating the data by institute and then by semester is kept in this step.

Finally, the cleanup script takes the files under `raw_data/participation` and changes the lecturer name column. Since all lecturers are denoted as "Surname, Name", but the rest of the data uses "Name Surname", the participation names are split at the comma, reversed and joined by a whitespace, to enable interaction between these files.

To simplify interaction with the cleaned data, the script saves all files in the new encoding of "UTF-8" to conserve umlauts and changes the column separator from a semicolon (;) to a comma (,).

4.4. The Old Report

To ensure the correctness of the provided data and to create a Minimum Viable Product (MVP), the first step was to recreate the manually generated current evaluation report with code and to compare the output with the existing reports. This section also aims to explain differences in results.

4.4.1. Implementation

The values provided in the table of evaluation reports (cf. Figure 1) are heavily summarized, which in turn simplifies the generation of the report. Most columns are created by simply summing up the related rows that satisfy the requirements in the rows and columns. The last two columns are easily generated by using the *median* function provided by `pandas`.

The first three columns need the participation data provided by the Advisor for Studies and teaching, which was already cleaned as mentioned in subsection 4.1. Here the number of registered courses simply is the size of the *DataFrame* and the collegial held courses and lectures with less than five registered students are extracted from the corresponding columns in the file.

Generating the remaining columns of the evaluation report requires the evaluation data. This data is split into the three institutes for computer science, mathematics and bioinformatics by default. To simplify working with the data it will first be combined into a single DataFrame containing all the data from all three institutes. During the merge every dataset will get a column called ”institute” appended, so the single evaluation responses can still be mapped to the respective institute. After that the responses are grouped into the courses they evaluate. This is done by using four columns that contain the course number, lecturer name, type of course (lecture, seminar or tutorial) and the name of the course. The number of responses per course is then appended as a *size* column, by counting the rows belonging to the group.

The four attributes mentioned above are the minimum set that can uniquely identify any student response to any course. While course numbers are unique for the courses, they fail to identify the ones held by two or more people that are evaluated differently. In order to differentiate between the people, the lecturer name is needed. In some cases lectures have the same course number as their tutorials and can be held by the same person, while still being evaluated separately, thus requiring to take course type into account. Finally, lecturers holding tutorials belonging to a lecture can lead multiple instances of tutorials, that are evaluated individually. In order to prevent the grouping of two tutorials held by the same person, the lecture name get's taken into account, since they are mostly different from tutorial to tutorial.

To get the number of courses with at least 5 student responses, the grouped courses are filtered by the newly created size attribute.

Calculating the percentage of courses that got more than five responses uses the number of courses provided in the participation file and the data generated in the previous step to calculate the resulting percentage.

Finally, getting the median for returned questionnaires and registered students in evaluated courses takes advantage of the *median* function in `pandas`. The function gets applied to the courses with at least five responses.

4.4.2. Differences in Generated Results

When comparing pipeline generated results to the official evaluation reports, it becomes apparent that numbers generated by the pipeline sometimes differ from the ones provided by the official report.

The biggest potential source of this data variation could arise during the grouping of the courses, since there is no way to uniquely identify a tutorial for a specific module. While the course name and number in combination with the lecturer name enables to uniquely group together tutorials, it is still possible that one person holds multiple tutorials with the same name. Since these tutorials share a name, lecturer and course number, they are grouped into a single tutorial, thus combining the results of the two tutorials. Currently, this does not happen often, but can potentially cause the loss of about half the data, since most tutors lead at least two tutorials per module.

Evaluation reports occasionally contain mathematical errors or differ in the way they calculate values. The report states it calculates the median of returned questionnaires per evaluated course with the restraint that the course has received at least five responses. On occasion the median is calculated considering all courses that got responses, even if they received less than five of them. The code only uses courses with at least five responses, thus reporting a different median. Similarly, the percentage of evaluated courses with at least five responses is calculated differently depending on the evaluation report. Each report takes the amount of courses with five responses as the percentage value, but some reports use the number of evaluated courses as the base value, while other reports use the number of registered courses.

4.5. The Question and Score API

In order to future-proof the pipeline it needs to be dynamic. This means it should be able to handle any question introduced in the future, but also needs to work if questions get dropped from the questionnaire. To implement these features, an additional API was created during development. This API is separated into two categories, which serve different purposes.

The first category is dedicated to questions. Here each question on the questionnaires is stored. Every question has a unique identifier that can be used to retrieve its content, scale and what dimension it belongs to. The API allows to either retrieve all questions or to get specific information about a question when provided with its identifier. In addition to the questions, this category also allows to retrieve all dimensions, course types and/or scales. A list of all endpoints is provided in subsection C.1.

This part of the API currently covers 74 out of 127 questions from every questionnaire from the last five semesters. Currently excluded are questions that aren't used anymore or that are variations of existing questions that haven't been identified yet. They can be found in Appendix D

The second category of the API was build to store the evaluation results. This category has multiple endpoints for retrieving, inserting or deleting evaluation results. It is possible to insert multiple courses at once with their respective results or to retrieve all of them. If scores are only needed for specific semesters, institutes, dimensions or all three at the same time, they can be retrieved by corresponding API calls. All endpoints are listed in subsection C.2.

The API instance itself gets delivered as two `docker`[4] containers that can be started together with recommended configurations using the provided *docker compose* specification. `Docker` is a tool that enables to create virtual machines (containers) that host all files required to run software that is by default isolated, preventing access to the underlying software, if not specified otherwise. It can be used on any system to recreate services identically, since containers get delivered with a description (called images) that tells `docker` what dependencies it should install with which version. This can guarantee that, regardless of their operating system, the user can execute software the way it was intended [6].

The first of the two containers simply runs a `Postgres`[5] Database image, which gets fed a `Structured Query Language` file that instructs the database what tables to create and what data (like current questions) to insert into them on startup. This file can be found in the public repository (see Appendix A). The second container hosts the actual code for the API. While both containers could be started independently of the compose file with their own provided *Dockerfiles*, it is not recommended. The compose file provides additional constraints that ensure full functionality. It creates a shared network that prevents the database container from being publicly accessible, thus only allowing the container housing the code to communicate with the database, securing it from attacks. Additionally, the compose file provides a health check for the database container, which causes the API container to wait with its startup until the database is ready. This prevents the API from crashing, since it tries to connect to the database upon startup.

The API was implemented in the programming language `JavaScript` using the `OpenAPI` specification. `OpenAPI` provides a standard to create API endpoints that are both easily readable by humans and machines [19], by providing standardized definition files alongside the actual API, that denote the behavior of the defined endpoints [20].

To follow this specification the API was developed with the `express-openapi`[6] package, which automatically generates `OpenAPI` 3.1.0 compliant APIs based on file structure. The package expects two folders, `services` and `paths` alongside an `api-doc.js` file. The API Document contains the definition of all types (here called models) that exist in the context of the API. Models typically provide their data type alongside with a short description and/or examples of input values. This file, is fully compliant to `OpenAPI` schemas[7] defining models.

The `paths` folder describes endpoints by following the folder structure to construct the URL. For example constructing the endpoint `/scores` the folder structure would look

[4]https://www.docker.com
[5]http://www.postgresql.org
[6]https://www.npmjs.com/package/express-openapi
[7]https://github.com/OAI/OpenAPI-Specification/blob/main/versions/3.0.3.md#schemaObject

like Figure 5.

```
/paths
└─/score.js
```

Figure 5: Directory tree for the `/scores` endpoint

In this example the `score.js` file hosts all logic for the endpoint. It defines what HTTP requests (GET, POST, DELETE, etc.) the endpoint accepts and how the response looks like. During development, it was decided that these files reference their respective service that handles the actual logic, while `score.js` itself would just call the service with the input data and return the response from the service.

In order to create endpoints that accept variable parameters, the parts in the path simply need to be enclosed with curly braces. Any part of the path can become a parameter, regardless if it is a folder or file. An endpoint like `/scores/{semester}/{dimension}` would allow users to replace *semester* and *dimension* with the respective values they want to request. The file structure for such an endpoint would look like Figure 6.

```
/paths
└─/scores
   └─/{semester}
      └─/{dimension}.js
```

Figure 6: Folder structure for the `/scores/{semester}/{dimension}` endpoint

The parameters delivered would be provided through the request parameters in the function signatures like Listing 1

```
async function GET(req, res, _next) {
    res.status(200).json(await service.function(req.params.semester, req.params.dimension))
}
```

Listing 1: Function signature for endpoints with parameters

As mentioned above, the endpoints reference services, which are stored in the `services` folder. These services contain functions that get called by the endpoint files. Typically, functions in the service files contain a common topic. For example everything that returns questions is handled by the `questionsService.js` file. If these services need to retrieve information from the database, they will call a function from the dedicated database file `postgres.js`, which actually communicates with the database. Functions in this file are exclusively syntactic sugar to hide SQL statements. These functions will convert the input, perform SQL statements to the database and return the output, while formatting in- and output it if needed.

To ensure all endpoints can be used by people without knowledge of the `OpenAPI` specifications, the `swagger-ui`[8] package was employed. It takes the provided definitions and endpoints and converts them into a graphical user interface that can be accessed in a browser. This user interface demonstrates descriptions for endpoints, what data they expect and return. It additionally provides the tools to make API requests directly. In the pipeline however, the API gets accessed by the `python requests`[9] package that calls the endpoints.

Creating the API this way not only allows code interacting with it to autodiscover endpoints through the standardized `OpenAPI` description, but also simplifies development of new endpoints or renaming/maintaining existing ones by following the mentioned file structure.

With both of these categories the API enables easy creation of new questions by inserting them along with their scales and dimensions into the database. The pipeline will then be able to recognize these questions in the raw data and convert them into the identifier (as described in subsection 4.3) to consider the student answers in the score calculation.

The API also implements the only "Could Have" that was put on the *MoSCoW* scale (see Figure 4), indicating that the feature was categorized falsely and actually is tightly-knit together with the "Must Have" of a dynamic pipeline.

4.6. The New Process

To enable the implementation of the mockups created and verified in the second phase of prototype design and interviews (see subsection 3.2.2) the transforming of answers to scores for the respective dimension is the focus of the new pipeline. Calculating the score for questions on a course basis provides the starting point for the scatter-, box- and violin plots (visualization see Appendix B). To calculate the score, every possible answer to a question has been given a value, which can either be positive, negative or neutral, based on what the answer conveys. Most answer possibilities are valued symmetrically along their scale. Due to the nature of Likert Scales, the possibilities have a neutral answer in the middle and get more negative/positive to the right/left of the neutral choice [3]. However, some answers were deemed "valueless". Specifically questions that asked how much time students spend on the course outside regular events, were seen as too individual to affect the score of the course. These answers were assigned a weight of zero and won't affect the score. Every answer with its weight can be found listed in the SQL file under `src/db/schema.sql` in the `GitLab` repository (see Appendix A).

To achieve this transformation from the Likert Scale into the new scores, the pipeline iterates over every file in the cleaned data and converts them into `pandas` *DataFrames*. In each *DataFrame* the student responses then get grouped by the course number, course

[8]https://www.npmjs.com/package/swagger-ui
[9]https://pypi.org/project/requests/

name, course type and lecturer name, creating groups for each individual course. Afterwards, the score function gets applied on each column of the group that matches a regular expression for UUIDs. The function takes the answer scale corresponding to the question and then maps the given student scores to the value of the answer in the scale. Once every answer to the question has been converted, the function returns the sum of all scores, creating the total score for the question while additionally obfuscating the individual responses.

Finally, these scores along with the course number, name, type, lecturer name, semester, institute, number of returned evaluations and number of registered students are put into the result API, which in turn inserts them into the database where they become a course object. The course gets assigned a UUID, which serves as a foreign key in a table that keeps track of scores, which also connects the question ID with an instance in the question table.

Courses may be identified uniquely by their ID, but to prevent course duplication, the database has an additional constraint. Any course in the database has an additional superkey, meaning the smallest number of columns that allow unique identification of any row in the database [1]. The course superkey consists of the five columns representing the course number, name, type, lecturer and semester. This means that courses that are repeated every semester are also created every semester. While any course can be uniquely identified by the first four mentioned columns alone, the semester column is still needed to fulfill one of the most mentioned Must Haves (cf. Figure 4). Keeping track of the semester for every course enables to keep track of historic development when working with the result data, since every iteration of any course keeps track of its own scores. Saving the history this way allows simplified queries by the API. With a simple select providing a course name, number, type and lecturer, the code can generate the historic development, without the need of performing complicated join statements in SQL. Additionally, the Advisor of Studies and Teaching mentioned that around every five years they have to delete the evaluation data. The semester column can be used to delete every course containing a specific semester without deleting the same course and by association the scores of a later iteration. Since the database also cascades deletions to foreign keys, the tracked scores for the specific course will also be deleted automatically (cf. Figure 7).

4.7. An Example Dashboard

In order to demonstrate how the data in the API can be used, the `/dashboard/dashboard.py` file contains multiple examples on how to modify data in order to present it using the mockups created and validated during subsection 3.2.2. Here the `python` solution for building dashboards - `streamlit`[10] - is used to generate graphs out of `pandas` *DataFrames*. This file is a ***demonstration*** and in no way production ready. It does

[10]https://streamlit.io

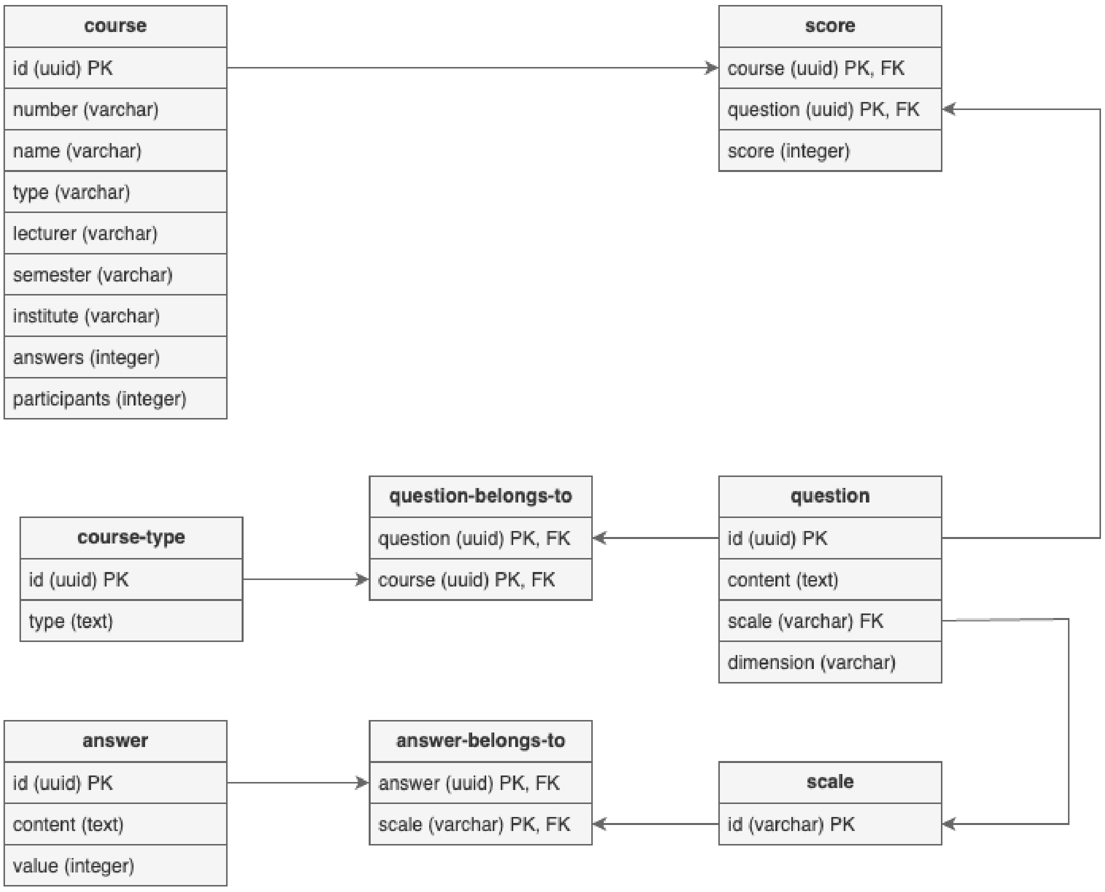


Figure 7: Relationships between tables in the Database

however provide multiple wrapper functions to generate charts interactively. Almost every chart has selection tools to filter data by institute, dimension or semester. It provides an example on how a single *DataFrame* can be used to generate almost every graph, by using `pandas` functions and the `python` plotting library `altair`[11]. The starting point is a *DataFrame* that contains all the data from the results API.

Initially scores for questions get converted to percentages for all dimensions using the equation

$$\forall question \in dimension : ((\sum question\_score) + (\sum max\_score))/((\sum max\_score) * 2)$$

where *dimension* is any dimension out of $\{A, B, C, A1, A2, B1, B2, C1, C2\}$, *question_score* is the score the student gave to the question and *max_score* is the highest score possible for the question multiplied with the amount of students that evaluated the course. Both sides of the division are added with the $\sum max\_score$ to move all answers into

[11]https://altair-viz.github.io

positive integers, thus simplifying the calculation of the percentage without falsifying the result.

These scores then can be used directly for the scatter plot where values from subdimensions (all dimensions ending with 1 or 2) are used as x and y values to place the courses. Filtering for the dimensions, semesters or institutes then is done by `altair` functions. Semesters are filtered using `transform_filter(alt.FieldOneOfPredicate( field="Semester", oneOf=semester_range))`, where `semester_range` is the range of semesters chosen by the user. Dimensions are encoded by changing what column the x and y-axis should represent.

For the box- and violin plots, the percentages are converted into seven intervals (each interval is $\frac{1}{7}$ of 100%), thus returning to a Likert Scale. The converted data then is used to generate the plots.

Since the line- and bar chart use participation data instead of scores, the initial data is used to group courses by semester and institute. Participation then is calculated for each course by dividing the number of responses with the number of registered students per course, which in turn gets visualized in the line- and bar chart.

The final chart is a table, which visually represents the historic development of participation for every semester in the data. The table highlights the current percentage while indicating with an arrow how this percentage changed since the beginning of the recording. Arrows are accompanied by a percentage indicating how much the value changed. Additionally, they are colored like a traffic light, where problematic developments are rendered red. This dataset does differentiate between course types, while the line- and bar chart won't. Thus, a new dataset is generated which groups courses by semester, institute and course category, but the participation is calculated the same way as before.

All examples are provided in Appendix E, additionally a more detailed explanation about how to organize and execute the pipeline can be found in the wiki provided in the `GitLab` repository in Appendix A.

Finally, the dashboard creates a ranking of the best courses per semester for the deanery evaluation. This ranking is created by summing all scores and dividing them by number of respondents, thus creating the average points per question.

In every chart except the last one the data is always created in a way that drops any values that could identify courses in the visualization. This fulfills the *anonymity* Must Have requirement from the *MoSCoW* prioritization (see Figure 4). The ranking can also be toggled if the dashboard should hide all personal data.

4.8. Assessing Reproducibility

During subsection 2.3 constraints for reproducibility and how they are realized have been introduced. While provding a license and hosting the code publicly accessible can be easily realized, creating reproducible code is not. As mentioned before, `Jupyter Notebooks` provide a useful tool to create code that promotes reproducibility. To validate the code, metrics identified by Pimentel et al. [13] were checked against the produced notebooks. In their publication, they analyzed 1,159,166 notebooks from 264,023 `GitHub` repositories in order to identify eight best practices notebooks should follow. The first practice dictates that a notebook name should be short and only consist of alphanumeric characters in combinations with a dot (.), underscore (_) and/or minus (-), which also applies to markdown headings. To ensure notebooks are verified as usable, before committing them into the repository, one should execute all cells in a sequential order from top to bottom. To ensure the provided dependencies are complete, they should be installed into a fresh environment and then be used to run the code. For file structure, they suggest putting imports into the first code cell in the notebook, check the end of the notebook for dangling cells not used anymore and to modularize code by using functions, classes or modules. Finally to ensure that the code can run on every system, imported files should be included by using relative paths, in order to be able to access these files on every system.

Most of these practices can be verified by using `pynblint`[12], which is a linter for `Jupyter Notebooks` that can check if notebook names follow a specific schema, if notebooks were executed linear, if cells exist in the notebook that weren't executed or are empty and if imports are at the top of the file [16]. By using this tool on all `Jupyter Notebooks`, most of the best practices mentioned above, can be verified. Currently, all notebooks pass the linting, on all rules, except the `cell-too-long` rule, which aims to keep code cells to a maximum of 30 lines. However, the cells failing this check either contain most function definitions or code that would be obfuscated if it would be modularized.

Additionally, all three notebooks developed during this thesis abstract code into single cells as much as possible, even providing a single cell dedicated for functions. The newest version also has been tested on a clean environment to validate its functionality. Finally, all file imports are done by dynamically building paths with the `os` module, without hardcoding paths, to enable file access on all systems.

On normal `python` files the python linter `flake8`[13] is employed. `Flake8` is a tool implementing various python coding guidelines, like the Python Enhancements Proposals, which are community driven code suggestions about coding conventions [14, 15]. Using this tool on normal python files provides sufficient information about reproducibility, especially since it's only used on supplementary files, which support the pipeline.

[12]https://pynblint.readthedocs.io/en/latest/index.html
[13]https://flake8.pycqa.org

A similar process is used with the `JavaScript` files used in the API (see subsection 4.5). Here the linter `Prettier`[14] is used to help make code more readable.

In order to ensure these guidelines are followed, `GitLab` executes and checks against them after every commit.

[14] https://prettier.io

5. Discussion

Once the second interviews were concluded, a second iteration of the UCSD steps of "Produce design solutions to meet user requirements" and "Evaluate the design solutions against requirements" (cf Figure 2) started. This thesis only started this iteration by producing a solution based on the requirements (see section 4), but did not conduct final interviews with the stakeholders to evaluate the design again to prove whether the solution meets the user requirements. However, since the users in the first iteration validated the dimension system and the forms of visualization, the argument can be made that the design was already evaluated and meets requirements, while the final color schemes and layout are not as important. It would still be recommended to show the final dashboard to the users again, collect feedback and build a solution based on it.

With the help of the API a user can always create a historic comparison of courses and visually represent this development without much effort as shown in subsection 4.7. While the dashboard may not be the most optimal visual solution, it definitely shows the potential of the different visualization possibilities the data has.

The overarching goal of this thesis was to help the evaluation committee in their work in the long run. This thesis provides the tools and collected the necessary input to start this journey. However, most of the work still stays within the committee. They still need to manually filter through data every year and identify meaningful statements from it. While the pipeline may automatically unify and normalize data, it still has caveats. As mentioned in subsection 4.4.2, the pipeline has data loss it can't recover. It now is up to the committee to develop a schema for their raw data that prevents data loss by machine interpreted results. Based on the requirement elicitation and the validation that followed, the faculty proposed needs that can be met using the pipeline.

5.1. Limitations

Ideally, reproducibility follows an *Open Science* approach, where not only results and the code are public, but also the initial data is publicly available [21]. Evaluation data is however, sensitive, so publishing it is impossible. Keeping this data private also obstructs any further individual research that could be using this data as a base, thus impeding on advancement of any supplements for the evaluation committee. This thesis aims to lower this barrier by providing example data in the repository (see Appendix A), that at least describes how the data looks. Creating an approach to anonymize data, without loosing meaning, can help the faculty in the future.

During the development of the pipeline, data was missing. Some semesters didn't have data for all institutes, which created gaps in the visualization. Especially missing or incomplete participation data eliminated some data points. If the pipeline is unable to find the number of participants for a course, it defaults to -1. This creates problems in all visualizations that use participation in some form, because courses that are missing this datum are excluded from the calculations. In order to unlock the full potential of the pipeline, the missing data should be added to the existing data.

5.2. Future

During subsection 3.2.1 a few interviewees mentioned that they would like to see motifs in the text responses by students that occur in multiple courses that should be highlighted. They declared that this can be positive but also negative feedback. The free text answers from students give them a tool that breaks the boundaries of the employed Likert Scale and gives them the freedom to express their personal grievances and likings with the course. If specific motifs are mentioned across multiple courses, the problem could be located outside the area of possibilities of the person teaching the course and may show problems with the study regulations or the institute as a whole. This can provide specific starting points for conversations with the student body on how these problems can be tackled. Implementing a system like that was beyond the scope of this thesis, as it would need a Large Language Model (LLM) to interpret the text answers and summarize them. The training and fine-tuning of such a model would be a bachelor's thesis on its own.

The provided visualizations in the dashboard (see subsection 4.7) currently are split by dimension and institute, however interviewees did mention that they could imagine splitting courses by their level (bachelor and master), course choice (mandatory or voluntary courses) or their classification (career preparation or core subject) can provide better information, due to the fact, that these visualizations better pinpoint where courses perform well or not.

Almost all interviewees mentioned that the results of individual evaluations at the faculty once were fully public to the student body and the lecturers. This was changed at some point due to different regulations. However, all people interviewed expressed their desire to return to this openness about the results, since handling the results this way showed students that their answers mattered and initiated conversations between lecturers about their (different) results. The pipeline itself tries to respect these wishes by keeping identifying information for courses, but removing them in the dashboard.

If in the future these requirements become reality, the pipeline provides the ability to easily transition from an anonymous approach to a more open one. It should however be noted that the API in its current state should not be accessible publicly, since it has no security measurements. Implementing such measurements like authentication via usernames and passwords can be implemented on desired routes by using the currently disabled `_next` parameter provided on every endpoint, where a middleware can be inserted that enforces security on routes that can delete, update or insert data to prevent malicious requests.

5.3. Conclusion

The created pipeline with all its supplements is able to fulfill the requirements provided by interviewees in subsection 3.2.1. With the provided cleanup (see subsection 4.3) the creation of the old report becomes possible, while also providing opportunities to create new visualizations (see Appendix E).

This thesis was able to provide decision makers in the evaluation process with useful tools they can use to fulfill requirements from personnel at the institute. Using these results as groundwork, the process can be enhanced in the future, where the (historical) data provided can verify changes made to evaluations, but also to honor good lecturers and motivate others to go into discussions with their colleagues or students on how they can improve their own teaching. While there exists discontent about how evaluations are handled, they are generally considered the right tool to measure student satisfaction with the faculty.

References

[1] C. J. Date. Codd's First Relational Papers: A Critical Analysis. https://www.dcs.warwick.ac.uk/~hugh/TTM/CJD-on-EFC's-First-Two-Papers.pdf, 2016. Accessed: May 2024.

[2] Agile Business Cosortium. Chapter 10: MoSCoW Prioritisation. https://www.agilebusiness.org/dsdm-project-framework/moscow-prioririsation.html, 2014. Accessed: May 2024.

[3] Bertram, D. Likert Scales. Tech. rep., Poincare, 2006. http://pages.cpsc.ucalgary.ca/~saul/wiki/uploads/CPSC681/topic-dane-likert.pdf, Accessed: December 2023.

[4] Blubbshoft GmbH. Evaluationen von Lehrveranstaltungen - umfassend und einfach. https://blubbsoft.de/Evaluation/Evaluationsprozess. Accessed: May 2024.

[5] Brandl, K., Mandel, J., and Winegarden, B. Student evaluation team focus groups increase students' satisfaction with the overall course evaluation process. *Medical Education 51*, 2 (2017), 215–227.

[6] Docker Inc. Use containers to Build, Share and Run your applications. https://www.docker.com/resources/what-container/, 2024. Accessed: May 2024.

[7] Gulliksen, J., Göransson, B., Boivie, I., Blomkvist, S., Persson, J., and Cajander, Å. Key principles for user-centred systems design. *Behaviour & Information Technology 22*, 6 (Nov. 2003), 397–409.

[8] International Organization for Standardization. ISO 9241-210:2010. https://www.iso.org/standard/52075.html, 2010. Accessed: May 2024.

[9] Jahnke, I. *How to Foster Creativity in Technology Enhanced Learning?* Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 95–116.

[10] Kitzes, J., Turek, D., & Deniz, F., Ed. *The Practice of Reproducible Research: Case Studies and Lessons from the Data-Intensive Sciences.* University of California Press, Oakland, CA, USA, 2018.

[11] Maheshwari, Arushi and Agarwal, Kartik. Copyleft: Copying Done Right. *Indian JL & Pub. Pol'y 4* (2017), 24.

[12] Olmsted-Hawala, E. L., Murphy, E. D., Hawala, S., and Ashenfelter, K. T. Think-aloud protocols: a comparison of three think-aloud protocols for use in testing data-dissemination web sites for usability. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2010), Association for Computing Machinery, p. 2381–2390.

[13] Pimentel, J.F. and Murta, L. and Braganholo, V. and Freire, J. A large-scale study about quality and reproducibility of jupyter notebooks. *IEEE/ACM 16th International Conference on Mining Software Reprositories (MSR)* (2019), 507 – 517.

[14] Python Quality Authority. Error / Violation Codes — flake8 7.0.0 documentation. `https://flake8.pycqa.org/en/latest/user/error-codes.html`, 2023. Accessed: May 2024.

[15] Python Software Foundation. PEP 0 – Index of Python Enhancement Proposals (PEPs). `https://peps.python.org/`, 2000. Accessed: May 2024.

[16] Quaranta, L., Calefato, F., and Lanubile, F. Pynblint: a static analyzer for python jupyter notebooks. In *Proceedings of the 1st International Conference on AI Engineering: Software Engineering for AI* (New York, NY, USA, 2022), CAIN '22, Association for Computing Machinery, p. 48–49.

[17] Regulin, M. Richtlinie zur Durchführung von Lehrveranstaltungen am Fachbereich Mathematik und Informatik. `https://www.mi.fu-berlin.de/fb/qualitaet/PDFs/Evaluationskonzept_final.pdf`, June 2021. Accessed: December 2023.

[18] Scheinholtz, L. A., and Wilmont, I. Interview Patterns for Requirements Elicitation. In *Requirements Engineering: Foundation for Software Quality* (Berlin, Heidelberg, 2011), D. Berry and X. Franch, Eds., Springer Berlin Heidelberg, pp. 72–77.

[19] The Linux Foundation. New Collaborative Project to Extend Swagger Specification for Building Connected Applications and Services. `https://www.linuxfoundation.org/press/press-release/new-collaborative-project-to-extend-swagger-specification-for-building-connected-applications-and-services`, 2015. Accessed: May 2024.

[20] The Linux Foundation. What is OpenAPI? `https://www.openapis.org/what-is-openapi`, 2023. Accessed: May 2024.

[21] United Nations Educational, Scientific and Cultural Ogranization (UNESCO). Understanding open science. `https://unesdoc.unesco.org/ark:/48223/pf0000383323`, 2022. Accessed: May 2024.

[22] Vinicius Fulber-Garcia. Requirements: Functional vs Non-functional. `https://www.baeldung.com/cs/requirements-functional-vs-non-functional`, 2024. Accessed: May 2024.

[23] Wes McKinney. Data Structures for Statistical Computing in Python. In *Proceedings of the 9th Python in Science Conference* (2010), Stéfan van der Walt and Jarrod Millman, Ed., pp. 56 – 61.

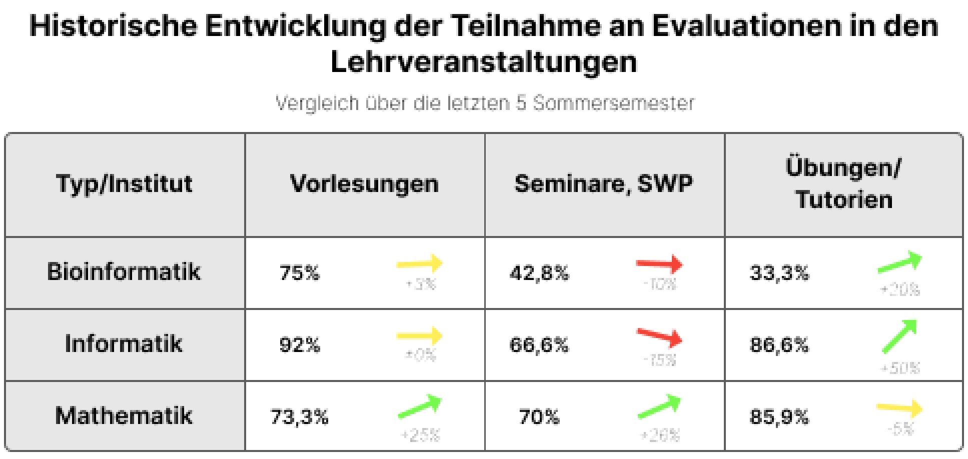
[24] Zentiks, S. R. Bericht zur Evaluation der Lehrveranstaltungen im Sommersemester 2023. `https://www.mi.fu-berlin.de/fb/qualitaet/Evaluationen/Evaluationsberichte/Lehrevaluationsbericht_SoSe2023_final.pdf`, 2023. Accessed: May 2024.

A. Git Repo

The GitLab Repository is publicly available under: https://git.imp.fu-berlin.de/alexander06/evaluation-report-generator

B. Mockups

B.1. Table

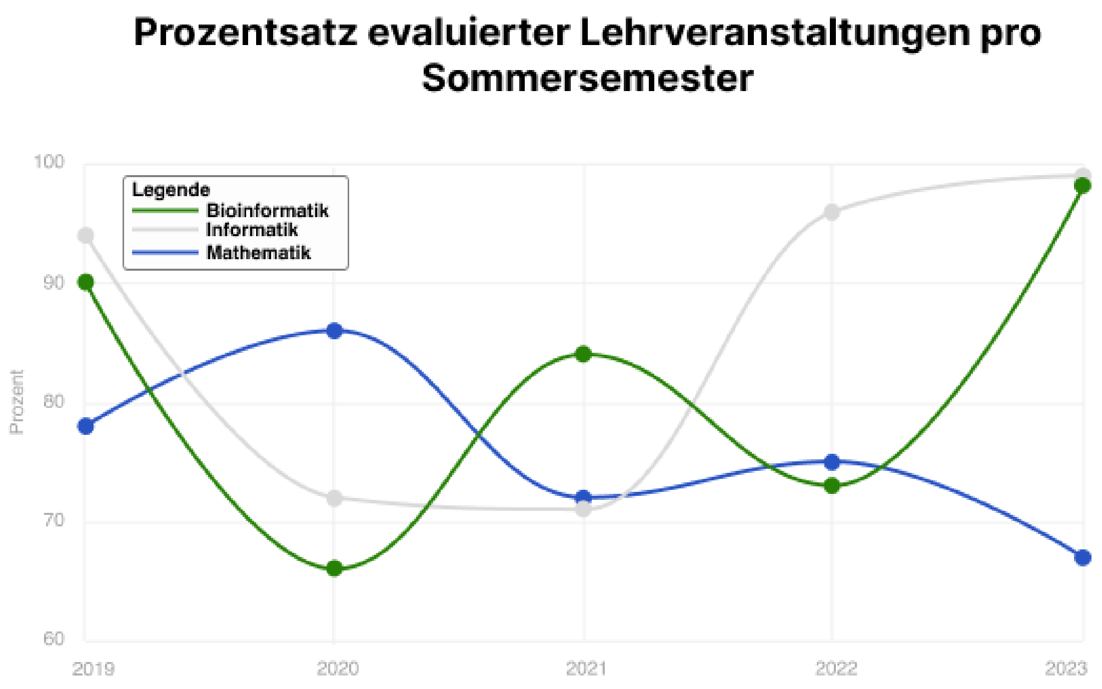


Historische Entwicklung der Teilnahme an Evaluationen in den Lehrveranstaltungen

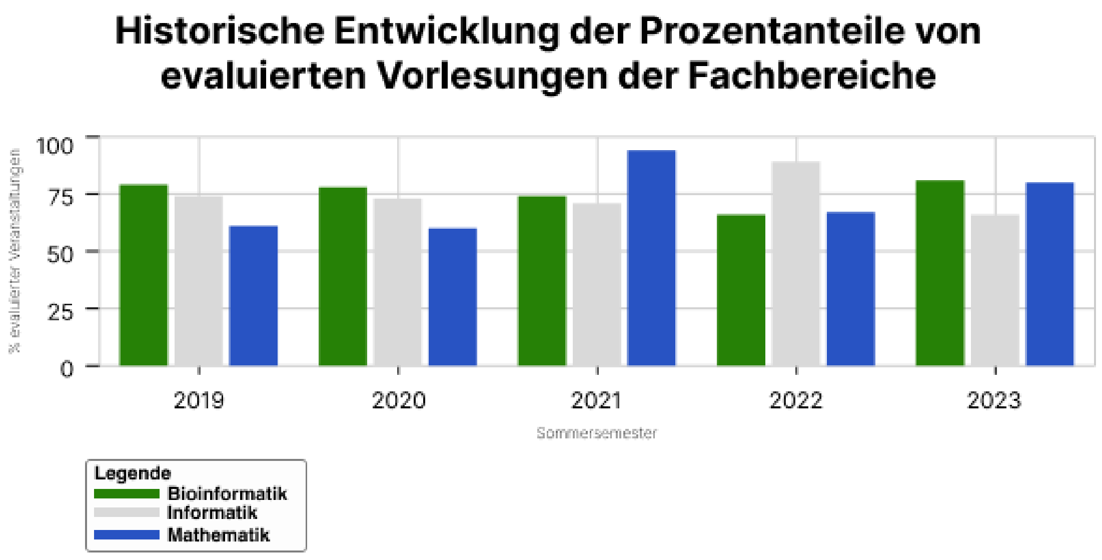
Vergleich über die letzten 5 Sommersemester

Typ/Institut	Vorlesungen	Seminare, SWP	Übungen/ Tutorien
Bioinformatik	75% → +3%	42,8% → -10%	33,3% ↗ +20%
Informatik	92% → ±0%	66,6% ↘ -15%	86,6% ↗ +50%
Mathematik	73,3% ↗ +25%	70% ↗ +26%	85,9% → -5%

B.2. Linechart

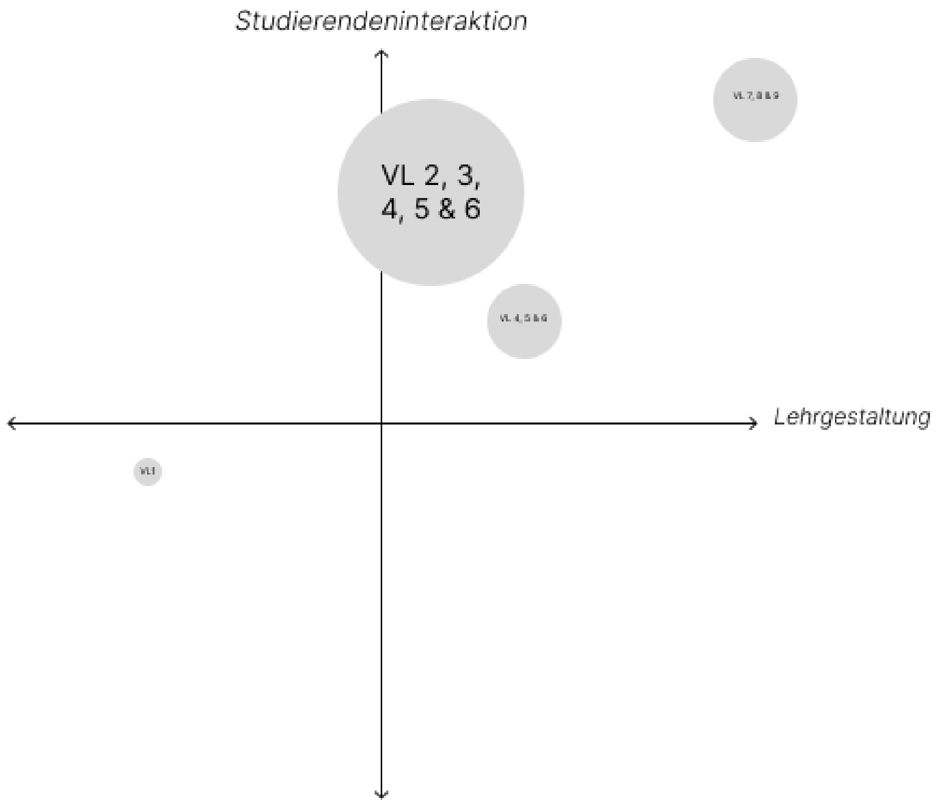


B.3. Barchart



B.4. Scatterplot

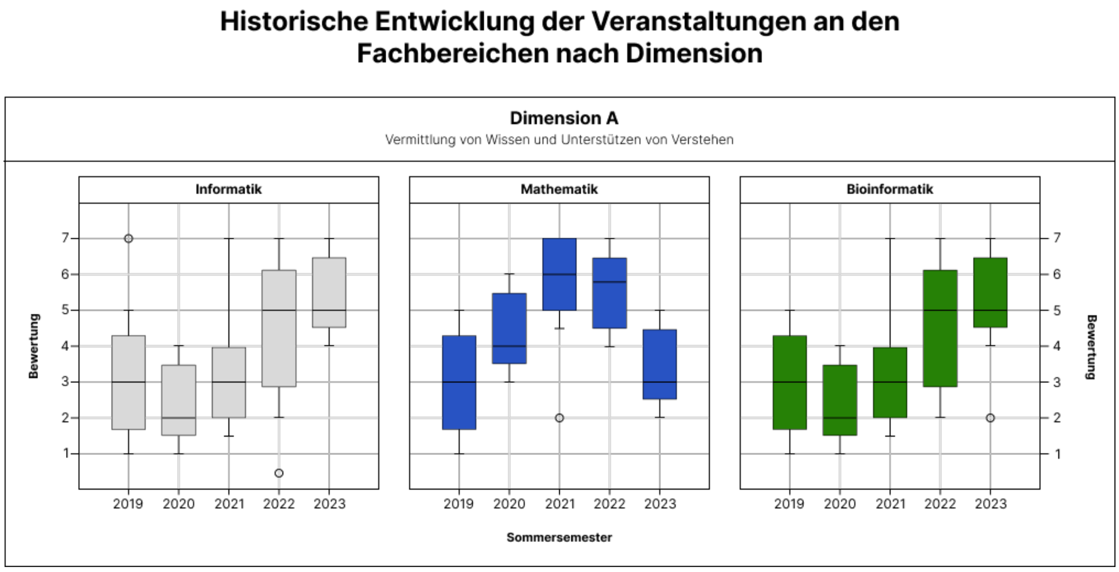
Clusterbildung von Veranstaltungen nach ihrer Bewertung in den Dimensionen



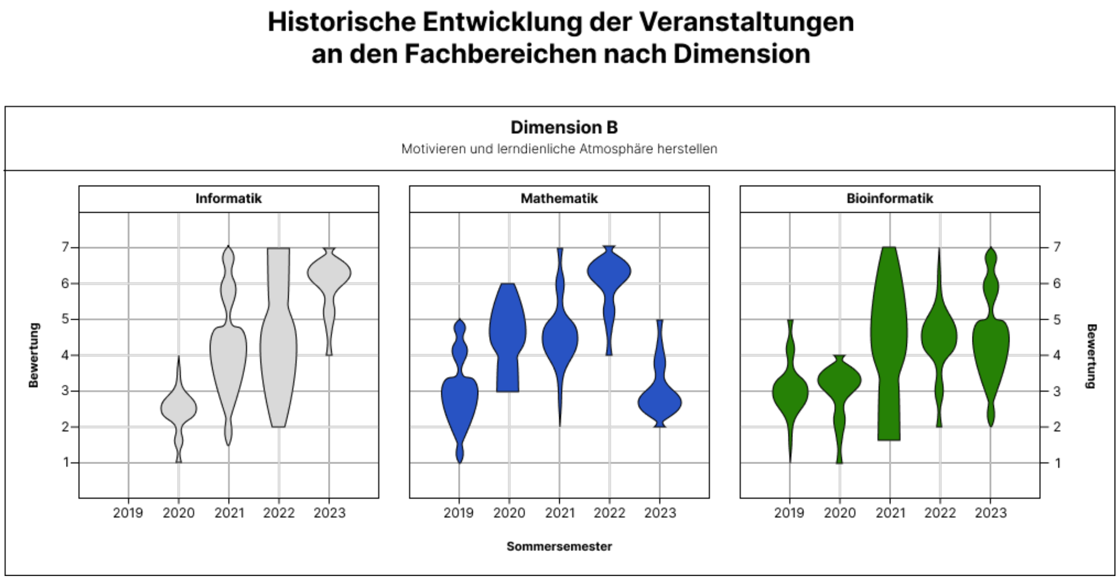
Dimension **Lehrgestaltung**:
Hierzu zählen alle Fragen der Evaluationsbögen, die die didaktische Gestaltung der Veranstaltung abfragen (Fragen über Mediennutzung, Verknüpfung von Inhalten und Tafelbild)

Dimension **Studierendeninteraktion**:
Hierzu zählen alle Fragen der Evaluationsbögen, die Einbindung von Studierenden abfragen (Fragen über Feedback zu Lösungen/Beiträgen, Erreichbarkeit der Lehrperson und Abfragen der Studierenden)

B.5. Boxplot

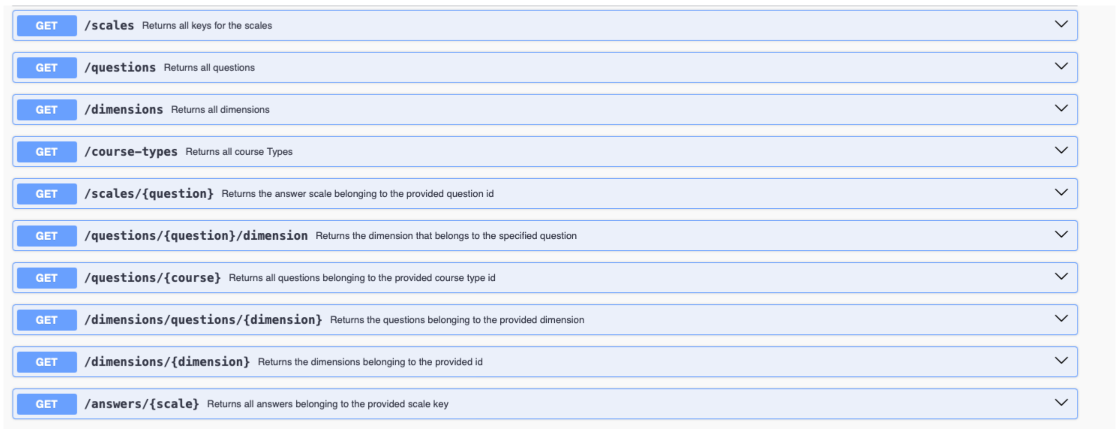


B.6. Violinplot



C. API Endpoints

C.1. questions



GET `/scales` Returns all keys for the scales

GET `/questions` Returns all questions

GET `/dimensions` Returns all dimensions

GET `/course-types` Returns all course Types

GET `/scales/{question}` Returns the answer scale belonging to the provided question id

GET `/questions/{question}/dimension` Returns the dimension that belongs to the specified question

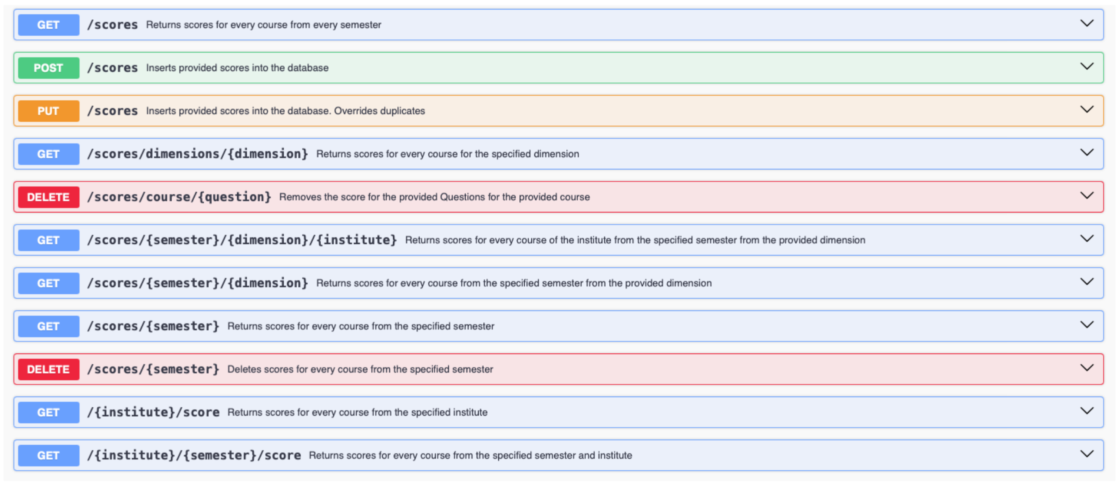
GET `/questions/{course}` Returns all questions belonging to the provided course type id

GET `/dimensions/questions/{dimension}` Returns the questions belonging to the provided dimension

GET `/dimensions/{dimension}` Returns the dimensions belonging to the provided id

GET `/answers/{scale}` Returns all answers belonging to the provided scale key

C.2. scores



GET `/scores` Returns scores for every course from every semester

POST `/scores` Inserts provided scores into the database

PUT `/scores` Inserts provided scores into the database. Overrides duplicates

GET `/scores/dimensions/{dimension}` Returns scores for every course for the specified dimension

DELETE `/scores/course/{question}` Removes the score for the provided Questions for the provided course

GET `/scores/{semester}/{dimension}/{institute}` Returns scores for every course of the institute from the specified semester from the provided dimension

GET `/scores/{semester}/{dimension}` Returns scores for every course from the specified semester from the provided dimension

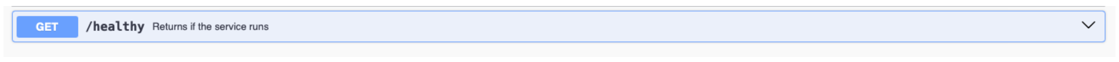
GET `/scores/{semester}` Returns scores for every course from the specified semester

DELETE `/scores/{semester}` Deletes scores for every course from the specified semester

GET `/{institute}/score` Returns scores for every course from the specified institute

GET `/{institute}/{semester}/score` Returns scores for every course from the specified semester and institute

C.3. telemetry



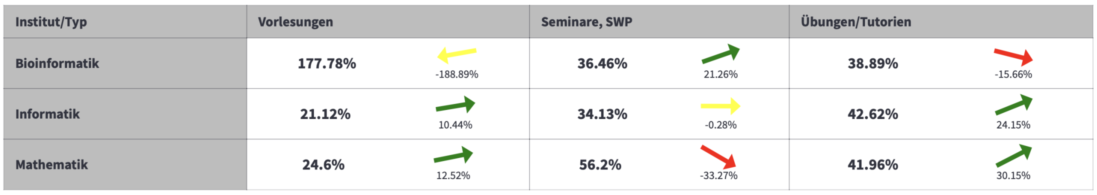
GET `/healthy` Returns if the service runs

D. Non-implemented Questions

```
['An wie vielen der angebotenen Live-Sitzungen haben Sie teilgenommen
↪  bzw. wie viele Inhalte der Lerneinheiten haben Sie bearbeitet',
 'Aus welchen Gründen haben Sie an der Hälfte oder mehr der
 ↪  Live-Sitzungen nicht teilgenommen bzw. nicht die Inhalte der
 ↪  Lerneinheiten bearbeitet?; Betreuung Kinder / Angehöriger',
 'Aus welchen Gründen haben Sie an der Hälfte oder mehr der
 ↪  Live-Sitzungen nicht teilgenommen bzw. nicht die Inhalte der
 ↪  Lerneinheiten bearbeitet?; Erkrankung',
 'Aus welchen Gründen haben Sie an der Hälfte oder mehr der
 ↪  Live-Sitzungen nicht teilgenommen bzw. nicht die Inhalte der
 ↪  Lerneinheiten bearbeitet?; Erwerbsarbeit',
 'Aus welchen Gründen haben Sie an der Hälfte oder mehr der
 ↪  Live-Sitzungen nicht teilgenommen bzw. nicht die Inhalte der
 ↪  Lerneinheiten bearbeitet?; zeitliche Überschneidung mit anderen
 ↪  Lehrveranstaltungen',
 'Aus welchen Gründen haben Sie an der Hälfte oder mehr der
 ↪  Live-Sitzungen nicht teilgenommen bzw. nicht die Inhalte der
 ↪  Lerneinheiten bearbeitet?; fehlende Motivation',
 'Aus welchen Gründen haben Sie an der Hälfte oder mehr der
 ↪  Live-Sitzungen nicht teilgenommen bzw. nicht die Inhalte der
 ↪  Lerneinheiten bearbeitet?; Lehrstoff selbständig erarbeitet',
 'Aus welchen Gründen haben Sie an der Hälfte oder mehr der
 ↪  Live-Sitzungen nicht teilgenommen bzw. nicht die Inhalte der
 ↪  Lerneinheiten bearbeitet?; technische Probleme',
 'Aus welchen Gründen haben Sie an der Hälfte oder mehr der
 ↪  Live-Sitzungen nicht teilgenommen bzw. nicht die Inhalte der
 ↪  Lerneinheiten bearbeitet?; fehlende technische Ausstattung',
 'Aus welchen Gründen haben Sie an der Hälfte oder mehr der
 ↪  Live-Sitzungen nicht teilgenommen bzw. nicht die Inhalte der
 ↪  Lerneinheiten bearbeitet?; keine Antwort',
 'anderer Grund/andere Gründe:',
 'Die Teilnahme an der Lehrveranstaltung hat mir Spaß gemacht',
 'Haben Sie sich (auch) unabhängig von der Lehrveranstaltung mit anderen
 ↪  Studierenden über die dort behandelten Themen ausgetauscht',
 'Beabsichtigen Sie (trotz der Aussetzung der Semesterzählung auf die
 ↪  Regelstudienzeit) die Lehrveranstaltung/das Modul dieses Semester
 ↪  erfolgreich abzuschließen',
 'Falls nein, nennen Sie bitte den Grund/die Gründe:',
 'Ich habe alle erforderlichen Informationen und Materialien zur
 ↪  Vorbereitung und Teilnahme an der Lehrveranstaltung rechtzeitig
 ↪  erhalten',
 'Das Gesamtkonzept der Lehrveranstaltung wurde zu Beginn ausreichend
  ↪  erläutert (u.a. Live-Anteile und selbstständiges Arbeiten mit
 ↪  digitalen Angeboten; verpflichtende und freiwillige Aufgaben;
 ↪  Termine und Fristen)',
 'Werden in dieser Lehrveranstaltung',
 '. präsentierte Inhalte (z.B. Folienpräsentation)',
 '. eigene Inhalte zu präsentieren (z.B. eine Referatspräsentation)',
 '. die Teilnehmenden im Rahmen der Live-Sitzungen mit Wortbeiträgen
 ↪  (Audioübertragung) zu beteiligen',
 '. gemeinsam an Inhalten zu arbeiten (z.B. mit Hilfe eines Whiteboards
 ↪  oder eines Wikis?)',
 '. in virtuellen Arbeitsgruppen in separaten Räumen zu arbeiten',
 '. den Einsatz von kurzen Umfragen oder Frage-Antwort-Formaten',
 '. bei Live-Sitzungen die Kommunikation über einen Chat',
 'das Arbeiten in studentischen Arbeitsgruppen (Untergruppe innerhalb
 ↪  einer Sitzung)',
 'die zur Verfügung gestellten Kursunterlagen (Skripte,
 ↪  Präsentationsfolien o.ä.)',
 'die als Video oder vertonte Folienpräsentation zur Verfügung gestellten
 ↪  Lehrvorträge',
 'die als Audiodatei zur Verfügung gestellten Lehrvorträge',
 'die zur Verfügung gestellten Videotutorials',
 'die zur Verfügung gestellten Selbsttests/interaktiven Übungen',
 'das/die in dieser Lehrveranstaltung genutzte/n Forum/Foren',
 'den/die in dieser Lehrveranstaltung genutzten Blog/s',
 'die/den in dieser Veranstaltung genutzten Wiki/s',
 'Die/der Lehrende war durch die angebotenen Interaktionsmöglichkeiten
 ↪  (z.B. Foren, E-Mail, Telefon-/Online-Sprechstunde) gut erreichbar',
 'Im Vergleich mit einem festen wöchentlichen Präsenztermin fiel mir die
 ↪  Auseinandersetzung mit den Inhalten der Lehrveranstaltung in dieser
 ↪  rein digitalen Lehrveranstaltung',
 'Der Stoffumfang, der in der LV behandelt wird, ist für mich',
 'stellt zu Beginn einer Sitzung den Zusammenhang zur letzten Sitzung
 ↪  her',
 'regt die Studierenden dazu an, die Richtigkeit ihrer Beiträge/Antworten
 ↪  selbst zu überprüfen',
 'verdeutlicht die Lernziele zu Beginn jedes Veranstaltungstermins',
 'präsentiert die Lerninhalte/den Stoff stimmig und kohärent',
 'gibt den Studierenden konkrete Hinweise zur Verbesserung individueller
 ↪  Leistungen',
 'unterstützt gezielt einzelne Studierende oder Studierendengruppen, die
 ↪  einen besonderen Unterstützungsbedarf haben',
 'formuliert Anforderungen so, dass der Großteil der Studierenden sie
 ↪  auch erfüllen kann',
 'ist in der Lage, eine ruhige und ungestörte Lernsituation
 ↪ herzustellen',
 'geht angemessen mit Unruhe und Störungen um',
 'beweist bei Unruhe und Störungen das nötige Durchsetzungsvermögen',
 'führt nach Beiträgen einzelner Studierender, die vom Thema wegführen,
 ↪ geschickt wieder zum Thema zurück',
 'Das Gesamtkonzept der Lehrveranstaltung wurde zu Beginn ausreichend
 ↪ erläutert',
 'Die/der Lehrende ist durch die angebotenen Interaktionsmöglichkeiten
 ↪ (z.B. Foren, E-Mail, Telefon-/Online-Sprechstunde) gut erreichbar',
 'Findet die Lehrveranstaltung in Teilen oder komplett digital statt?
 ↪ (Papier-Evaluation: Wenn Sie "nein" angekreuzt haben, dann fahren
 ↪ Sie bitte mit "Erfassen des Outputs" fort.)',
 'Das Studieren in dieser digitalen Lehrveranstaltung fällt mir im
 ↪ Vergleich zu einer reinen Präsenzlehrveranstaltung',
 'Im Vergleich mit einem festen wöchentlichen Präsenztermin fällt mir die
 ↪ Auseinandersetzung mit den Inhalten der Lehrveranstaltung in dieser
 ↪ rein digitalen Lehrveranstaltung']
```

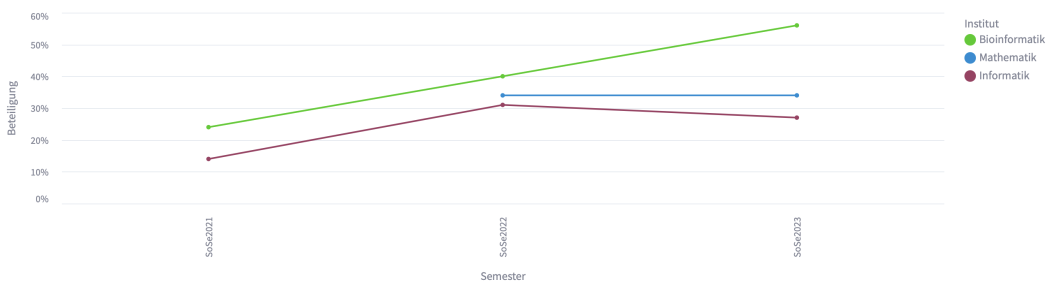
E. Dashboard

E.1. Table

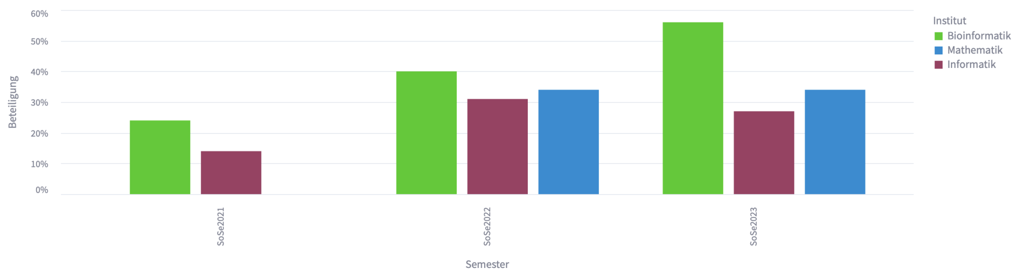


Institut/Typ	Vorlesungen		Seminare, SWP		Übungen/Tutorien	
Bioinformatik	177.78%	← -188.89%	36.46%	↗ 21.26%	38.89%	↘ -15.66%
Informatik	21.12%	↗ 10.44%	34.13%	→ -0.28%	42.62%	↗ 24.15%
Mathematik	24.6%	↗ 12.52%	56.2%	↘ -33.27%	41.96%	↗ 30.15%

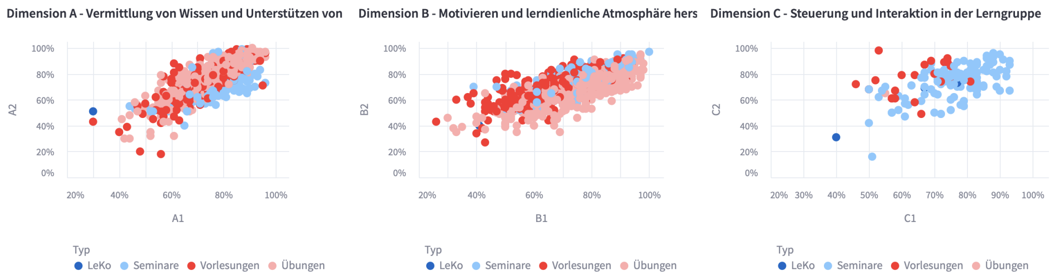
E.2. Linechart



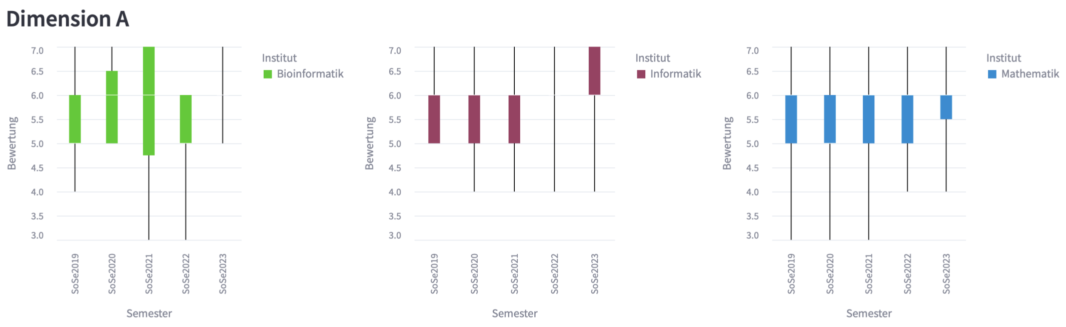
E.3. Barchart



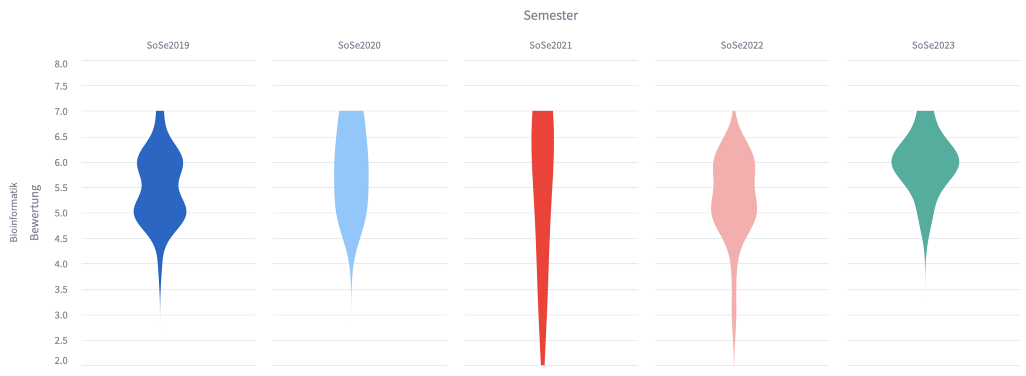
E.4. Scatterplot



E.5. Boxplot



E.6. Violinplot


Semester
SoSe2019
SoSe2020
SoSe2021
SoSe2022
SoSe2023
Bioinformatik
Bewertung