Interactive Data Visualizer

The Development of a Graphical User Interface for Data Science

Glenn van den Belt
Shaan Hossain
Joost Jansen
Adrian Kuiper
Philip Tempelman

TU Delft

Preface

This report was written by a group of five computer science students at the Delft University of Technology. For the final quarter of our second year, we created a graphical user interface that visualizes data onto graphs and tables. We designed this application for an external client for the Software Project course (CSE2000). The application is called the Interactive Data Visualizer and is developed for the researchers from the TU Delft research group Pattern Recognition & Bioinformatics.

While writing this report we assumed the reader to have a basic understanding of the software development process and the programming language Python. Readers particularly interested in the implemented features can find these in chapter 5. Software developers interested in the possible expansion of the application can find recommendations in chapter 8.

We would like to thank our Teaching Assistant R. Vogel and our coach F. A. Oliehoek for their constructive feedback on the development process. We are also very grateful for the challenges that our client has brought during the project. A special thanks to A. N. Jahfari for not only taking on the role of being our client, but also for providing us with feedback and for being open to all questions.

Delft, June 2021
Glenn van den Belt
Shaan Hossain
Joost Jansen
Adrian Kuiper
Philip Tempelman

Summary

Researchers from the TU Delft research group Pattern Recognition & Bioinformatics have been collecting data from heart rhythm disturbance patients. To draw conclusions from this data and find correlations, they needed a tool that helped visualize this data. Such a tool with minimal functionalities was created by them. This tool needed improvement and perhaps be completely redone. With the help of predetermined requirements provided by the researchers and existing frameworks, the Interactive Data Visualizer was developed.

This report documents the research, implementation, and usage of project assisting tools of this data visualizer. For research, the following question was answered: *"How can an interactive data visualizer be built using freely available tools and existing software?"*. The study concluded that the Dash framework was the best option to use in this case. The variety of already present features in this framework made it possible to implement the requirements. In this application, the user can upload one or multiple files containing data. Afterward, they must choose their x-axis and y-axis followed by the label they want to plot. Then features such as nested filtering and applying user-defined functions can be used. To complete the application and make it more applicable in general cases additional research was needed.

Firstly, research was done to discover what features other than those requested by the client should be present in the data visualizer. The application should not only be applicable for one specific case, that is data from heart rhythm disturbance patients, but should also be applicable for all other kinds of data. This resulted in the addition of features such as the support for multiple types of graphs. Ranging from scatter plots to density graphs. Another feature is the possibility for the user to plot multiple different graphs.

Secondly, research was done to determine what design choices should be made in order to make the application intuitive for the user. For this, multiple sessions with the client were needed to match the design to their preferences. Also, a literature study was required to discover what is considered intuitive for the average user. This resulted in an abstract design with the graphs in the center of the application and a collapsible sidebar with its respective buttons. Since the application is programmed in Dash, the application will run on a local server in any web browser.

The goal of creating this data visualizer was achieved by combining the best-suited framework, the previously mentioned features, and creating an intuitive design. For further improvements to the data visualizer it is recommended to port it to a web application, add the possibility to import and export configurations, and make it possible to visualize possible correlations in between the graphs.

Figure 1: User Interface of the Interactive Data Visualizer when using the Avocado Prices dataset [1]

Contents

1 Introduction

Every day, our society creates about 2.5 quintillion bytes of data, that is 2.5 followed by 18 zeros [2]. Data has also been dubbed the new oil and data scientist has been named the sexiest job of the 21st-century [3]. However, like with oil, data is meaningless when it is not processed properly. One way to process data is through visualization. Visualizing data might bring valuable correlations to light that could not be derived before. Nevertheless, looking at a visualized form of data and finding correlations is not easy. Yet in most cases, it is a lot easier than trying to derive correlations from just a table of numbers. However, before someone has the opportunity to do so, they need to visualize the data in a practical and meaningful way. Therefore they would need a sort of data visualization application.

To build this application various challenges need to be overcome. Ranging from practical challenges such as slow response times when dealing with large data sets to more functional challenges such as removing noise from data to make valuable information more clearly interpretable. Furthermore, since every data set is different, a one-for-all configuration would not be very practical. Therefore the application needs to be interactive. The described application enables its users to derive valuable correlations that could deem very helpful. For instance for researchers like our client, who is researching the causes of heart rhythm disturbances. Unfortunately, our client has found no application that satisfied all their needs and has therefore put up the assignment to build such an interactive data visualizer application.

This report answers the question of how an interactive data visualizer was built using freely available tools and existing software. To achieve this, the report documents the research, design, planning, implementation, and testing that was done before and during the building of the application. First, during the research phase, a lot of questions were asked to determine the exact requirements. This was followed by thorough research and testing of existing tools and software libraries. For each library, their functionality and extensibility were researched. They were then compared with each other to find the libraries that best suited the client's requirements. When it was clear what software and tools were available, the design and planning process was started. During the implementation phase, the design and planning were changed frequently. The team was often ahead of schedule and therefore more could be done than initially expected. The majority of the research was also done during implementation. For instance on certain library functionalities or good graphical user interface design. Also, either manual tests or written tests were used to test the performance of our application. Since the client wanted to be able to easily extend the application, the extensibility of the application was emphasized during the implementation. After most of the implementation was done, the application was tested by various random users. These users had varying backgrounds and had not seen the application before and received no extra instructions. Based on these experiences the application was evaluated and improved where necessary.

The report is structured as follows. Chapter 2 provides a problem analysis consisting of the problems that arise when working with large datasets. Followed by solutions on how they can be tackled and how some solutions are created with already existing software. Chapter 3 elicits the requirements in a MoSCoW method manner. Next, in chapter 4 ethical questions about the design of our product

are being raised and answered. In chapter 5 the final product will be presented. All its features are discussed in the subsections along with the reason for their presence. Chapter 6 gives an overview of the important assisting tools we used during our project. Chapter 7 reviews if the right product was build and the planning and teamwork of the team. Chapter 8 contains a discussion about the final product. It discusses the realization in terms of requirements and includes the future recommendations of the team. Chapter 9 mentions the purpose of this project, a reflection on how the team has worked together on the project, and a conclusion on whether the team has reached the goals they aimed to achieve with this project. Finally, Our appendixes consist of the requirements questions we asked our client, the team's schedule and the project plan.

2

Problem Analysis

This chapter aims to create an understanding of the problems that arise when visualizing data and lists its potential solutions. The common shortcomings in the present-day visualizers is analyzed in Section 2.1. The type of data the user can visualize and how they can interact with it is described in section 2.2. Section 2.3 describes the complications that big data sets pose to the visualization process. Section 2.4 elaborates on how the application can be built using an existing framework and how it should be extensible for future developers.

2.1. Shortcomings in existing data visualizers

When visualizing data the user can use a large variety of features in order to create their optimal graph. For the developers, it becomes a challenge to find out what these features might be. However modern-day visualizers lack several features which would be considered crucial according to our client.

When looking at the data visualizer from Juiceanalytics [4] the main purpose of the application is believed to be based around presenting the data rather than using multiple features. Features such as selecting data and plotting this subset were not possible. Without this possibility, it made it hard to disregard irrelevant data. Another shortcoming in the Juice application was the visual design of the application. The main purpose of the application, plotting a given set of data onto a graph, is not possible until many ambiguous options were selected. In contrast to easily interchangeable components and graphs, choosing the type of graph and giving the axes is a label was not a trivial task. The option to create graphs was hidden underneath a layer of settings and so-called sections. Conclusively, the features and the design were focused on visualizing data. Manipulating the data however is not possible. Applying filters is therefore not possible which according to the client is an important feature.

2.2. Data visualization

Some features are considered a high priority when building this application.

First of all, the type of data that will suit the application needs to be analyzed. The application must be built to be able to handle all sorts of data. Generally, this means that the data must be generalized before this is used. Fortunately, the client indicated they will format their data in such a way that it will be compatible with the application instead of the other way around. Since most data will be derived from values that were originally continuous, in some cases, it will be more natural to display them in a more continuous manner. This could, for instance, be done by providing the option to graph the data into a heat or contour map instead of the standard scatter plot. Furthermore, each data point carries its own metadata that must be visualized when hovered over it. This metadata could be provided either in a table separated from the original graph or directly visible within the graph.

Secondly, the request from the client was made to make it possible to filter the data multiple times. To be able to graph, for instance, the data of 'female patients born after 2017', the user must be able to provide certain filters. And because they might want to use multiple filters simultaneously, the application should be able to do nested filtering. When executing the nested filters, parsing the whole data set again must be avoided since this would be very memory intensive and, therefore, very inefficient.

Therefore a convenient way to store intermediate results must be found to ensure that the memory use is minimized. A potential solution to this could be to cache the intermediate results.

At last, another specific request from the client was the addition of thumbnails. Although graphs of many patients combined are very interesting, the user might also want to inspect data of an individual patient. Therefore our application should show a thumbnail of a graph of individual patient data whenever the user hovers over a data point of the corresponding patient in the original graph.

2.3. Computation intensity

As the client will provide a large amount of data, which could even be in the order of Gigabytes, it might be cumbersome to load all the data in at once. A possible solution could be to choose the order of the data to load. With this, the user can prioritize a certain part of the data and improve the loading time of the application. Another problem that might arise is the loading time of the graph. A set of data with millions of points will take a long time load. A possible solution could be a filter option. This option will make it possible to apply filters over the uploaded data. Its purpose is to decrease the number of data points the user wants to have visualized.

At last, extra attention should be paid to memory usage. If the application gets expanded to a web app this might become a problem. Because the user's browser might be limited in terms of memory. This will in a poorly functioning application and possibly unusable. Another consideration is to require the user to have a certain amount of memory available on their device. This will make the trade-off between performance and availability lean more towards performance-sided.

2.4. Application framework

The overall design of the back-end of the application will determine how extensible this application will be.

Firstly, the choice of a library for the application was an essential task. The choice consisted of using either an extensive, high-level library and incorporate its functionalities or using a low-level library and build all the functionalities from scratch. Most high-level libraries are not easily extendable in terms of features and are thus relatively limited to what we can do with them. However, depending on how rich the library is, using this solution prevents users from writing a lot of functions and algorithms themselves if they want additional features. In addition, this option is a lot less error-prone. The other option, using low-level libraries and building all the functionalities from scratch, might be the only viable option if there is no suitable high-level library. Low-level libraries give users more freedom, but using them will be more complicated since users will have to create everything from the ground up. Furthermore, using them also requires users to potentially have to combine multiple different libraries in order to achieve their goals, which might also be a complicated task.

Extensibility

Future developers must be able to easily extend the visualizer with additional charts, graphs, filters, or functionalities. An object-oriented framework that makes it straightforward to extend the visualizer with additional features is essential. According to G. Kakarontzas and I. Stamelos, the optimal approach to creating an extensible application is by using the Tactic-Driven Process. At first, the general model must be created where relevant dependencies between classes are visible. Secondly, test scenarios must be considered where an arbitrary developer would improve the application. At last, the previously mentioned test scenarios must be executed in order to determine if the model is sufficient for implementation [5].

Among thoughtfully designing the back-end of the application it must also be future-proof. Any function or attribute, added or altered must work coherently with the initial application. This means that the design must consist of components that are interchangeable and removable. Any future developer can thus change the looks without causing the application to malfunction or behave unexpectedly. Additionally adding new components must also work properly. The user adding their own function or algorithm should be possible but not cause the application to fail. It must thus be able to work with any well-defined function in order to run this function or algorithm properly.

3

Requirements Elicitation

In this chapter, the requirements for the Interactive Data Visualizer are discussed. The elicitation process and its findings are briefly described in sections 3.1 and 3.2 respectively. Lastly, in section 3.3, both the functional and non-functional requirements are listed using the MoSCoW method.

3.1. Process

At first, we thought about the stakeholders involved. Since the only stakeholder involved is our client, this made the requirement elicitation process relatively simple. In preparation for the meeting, we constructed a set of questions. These questions are based on the project description found on the TU Delft Project Forum website. Our questions were constructed in such a way that the needs of the client are portrayed clearly.

During the meeting with the client, we have noted the answers to the before mentioned questions. These questions and answers are listed in Appendix A. Besides answers to our questions, an oversimplified demo application [6] made by the client was provided.

3.2. Findings

The client indicated that they are working with very rich time-series data extracted from heart rhythms disturbance patients, such as information on step counting, heart rate bpm, and more. In addition, the client also indicated that handling this data is often a complicated task and that ”just putting everything into an algorithm” is often quite useless. Therefore, the client would like an application that visualizes the patients’ data interactively to be able to retrieve valuable information.

The data set sizes that are handled often extend multiple gigabytes in size. To efficiently manage data of this size, the client has suggested that the application should load the data lazily. In other words, only the data of one particular patient needs to be plotted. It will not be necessary for the application to load all the data of the patients at once. Lazy loading will make the application more memory-efficient, especially when only subsets of the entire data set have to be plotted.

Furthermore, the client requested a feature that enables users to perform nested filtering. For example, when plotting the data of all patients, a user should be able to select a set of patients and plot only the data from the years after 2010 for that specific patient set. The addition of this feature allows the user to reduce the size of the data they want to have plotted and view the data from a different angle.

3.3. MoSCoW method

After successfully forming a general description of the final application a more thorough and formal list of requirements was made. This was done to align our concept with that of the client. This is always a very important step in software engineering to prevent any misconceptions about the implementation. While listing these requirements the MoSCoW method was used. The MoSCoW method is a very famous

method that distinguishes the requirements into 'must', 'should', and 'could haves'. Furthermore, it also allows for non-functional requirements, as well as the 'will not haves'. In a case study of Hatton (2007) it was listed that on a scale of 1 to 10 the median difficulty and median confidence was respectfully a two and an eight. [7] Therefore the MoSCoW method is a very easy and efficient tool to use. A list of all of these requirements now follows.

Non-functional

The system must be...

- Able to select, filter and visualize data within 2 minutes.
- User-friendly to the extent that people from all machine learning backgrounds are capable of understanding how to use it within 15 minutes.
- Extensible to the extent that developers can understand how to extend the application within 2 hours.
- Able to handle data set sizes of up to at least 5 gigabytes.
- Built in Python 3.
- Able to run in Linux.

Must Have

The system must be able to...

- Select data using a selection method such as 'rectangle selection' or 'lasso selection' or something similar. a
- Plot data points with a scatter plot.
- Use multiple machine learning tools written by the user on the loaded data.
- Show the meta-data of a specific point in a thumbnail, which for example, pops up when hovering over that specific data point.
- Visualize data that has three or less dimensions.
- Run locally.
- Run on data written in a .csv file.
- The user must be able to filter on specific points.

Should Have

The system should be able to...

- Be extendable so that the client can implement additional data visualization and filtering methods.
- Have a short description in which the current tool of visualizing the data is explained.
- Plot continuous data with a heat map.
- Support injecting any functions before plotting. An example would be a function to compute a density map before plotting or plotting the trajectory of data over time by connecting data points with lines and arrows.
- Avoid parsing the whole data set again when applying multiple filters.
- Support nested filtering.
- Create multiple subplots from one large plot and view them in on single time series

The user should be able to...

- Give an order of loading when a data set is loaded.
- Export and download figures made by the system.

Could Have

The system could be able to...

- Run as a online web application.
- Load in a .csv file with meta-data within the desired table or with meta-data in a separate table.
- Make a selection of specific colors for specific data sets.

The user could be able too...

- Use a Drag & Drop to upload the data.
- Save presets of settings in the form of ticked boxes etc.
- Load in a .csv file and select, filter and visualizes the data.

Will Not Have

- Ability of users to have an account.
- Converter of data, since the client will deliver data in our demanded format.
- There will not be any support for any local/online databases since all data will be loaded in
- There will not be support for mobile and tablet versions.

4

Product Design and Ethical Implications

This chapter presents the design approach used in the project and the ethical implications of these design choices. Great responsibility is required when handling the medical data of patients. In section 4.1, it will become clear why our application's data is very sensitive and what the consequences for an individual could be when the medical data would be leaked. Section 4.2 explains how leaks are prevented in our project.

4.1. Sensitive data

This section elaborates on the data our application will be handling and why it is so important that we handle it responsibly.

Privacy issues

Heart rhythm data of patients are very sensitive since exposing the information would violate a patient's privacy. As stated in chapter 2, the visualizer deals with massive data sets on heart rhythm disturbance patients. The data contains aggregated data points of many patients and individual data of each patient over time. Whether aggregated or not, this data is still very sensitive, and everything must be done to prevent it from falling into the wrong hands. Other parties might like to have access to this data to, for example, determine health insurance fees or to conduct criminal activities like blackmailing. Chris Bowen, ClearDATA Chief Privacy and Security Officer Founder, says that someone's medical record is 50 times more valuable than their credit card number. "You can build an entire human persona around a health record, seek or create medical treatment, abuse drugs or get prescriptions." [8]

Exposing individuals

When heart rhythm (disturbance) data are leaked, particular individuals could experience harassment. Data might show a correlation between having your heart pump at a relatively high bpm and having a heart rhythm disturbance condition. One might then conclude that an individual has a higher chance of having such a condition when their heart beats relatively fast. Someone might deduce this before the one with the fast-beating heart knows about this correlation and before it is even known if there is a causal relation. In the best case, the one making the potentially wrongful deduction only lets the victim know and gives some tips on how to lower your heart rate. In the worth case, the victim gets harassed because the one deducing thinks the victim is responsible for their health insurance getting more expensive. Although this might not be the most likely scenario, one should not disregard it. Besides the fact that almost all harassment is wrong anyway, an individual might sometimes be unable to do anything about their high heart rate or heart rhythm disturbance condition. One could argue that since everyone knows smoking is bad for your health and gets little to no harassment, we should not fear this. However, this might be because smoking is pretty much engraved in our society, or because it is addictive and therefore harder to prevent, or for another reason. The fact remains that everything must be done to prevent this slight chance from being possible.

Ethical questions

Some questions now arise: *is it fair to make people with heart rhythm pay more for health insurance? And what about people with heart diseases that could not have been prevented but are just 'bad luck'?* These are questions that are very hard to answer, and there probably exists no perfect answer to them. According to Strech and Sofaer, experts in ethics and philosophy of medicine, a systematic review of reasons is needed, meaning viewing the ethical dilemma from all different perspectives to improve ethically relevant decisions [9].

A general look at this would be to say that people who smoke or who eat unhealthy food should pay more for their health insurance. On the other hand, what if those people didn't have a choice to eat healthier food because it's more expensive and they don't have the means to pay for it? These kinds of dilemmas make it very hard to say whether or not it is good practice to release these kinds of data publicly. However, deciding this is not up to the developers, so they should do everything in their power to prevent their application from forcibly making such a decision, for instance, by leaking all the data to third parties. Therefore developers must treat the data very carefully.

4.2. Prevention of leaks

According to Kim et al.[10] privacy of big data can be referred to three matters: data security, access control, and information security. Therefore we will focus on these three topics. To prevent leakage of patient data, the application will run locally and on local data. This will prevent any unwanted access control. Our application does not need to send or receive any data to and from online resources. Since, by far, most hacks and leaks occur over an internet connection, and our application doesn't need any internet connection, the risk of the data being compromised is greatly reduced.

Eventually, the application might be turned into a web application. This requirement is not an obligatory feature but rather a possible extension. If the application is converted to a web application, a lot of time and effort should be spent on making sure our application is impenetrable before it is deployed. At last, consulting an expert in the data security field and adjusting the application accordingly is recommended.

5 Implementation

In this chapter, the most important features of our final product are described to get an overview of the implementation of the application. In section 5.1 the essential functionality is discussed; plotting. We discuss how you can easily compare multiple graphs in section 5.2. Then, in section 5.3 it is described how our application supports uploading multiple files and how they can be used together. In section 5.4 it is documented how we implemented nested filtering. In section 5.5 it is discussed how data selection can be used and how we implemented it. The way we allow user-defined functions and why is elaborated on in section 5.6. And finally, in section 5.7 we describe how we designed the GUI and our motivation for doing so.

Table 5.1 gives a short overview of all the features that were implemented in the application.

Feature	Explanation	Reason
Plotting	See the data in a visual form, such as a graph	Data can be visualized in a graph
Multiple graphs	Multiple graphs can be plotted simultaneously	Data can be plotted onto multiple graphs can visualize multiple data sets
Multi-File support	Multiple data files can be uploaded and plotted	User can compare different sets of data with each other as the different sets of data can be viewed in multiple graphs
Nested Filtering	Data can be filtered before plotting and on the spot	User can filter data to desired subset and the loading time decreases when user filters data before plotting
Data Selection	Specific data points can be selected	User can use the selection to perform further analysis on this selection
User-Defined Functions	User can apply their own function to modify the data	User-defined functions like PCA, can be applied to perform accurate analysis
GUI design	The layout of the application	Design must be intuitive for a general user

Table 5.1: All features present in the Interactive Data Visualizer

5.1. Plotting

Plotting data is the essential feature of the application. The application has at least one main graph in which the user can select which data features to use for the axes. For this graph, the user can also select what feature to base the colors of the data points on. Furthermore, the user can select the plotting method, such as 'scatter' or 'density', which will result in a scatter-plot or density-graph, respectively. The possibility for these varieties of possibility is because various types of data suit different plotting methods. Besides the main graph, there is also an additional graph available to plot onto.

5.2. Multiple Graphs

Besides the main graph, additional graphs are present to plot onto. To compare different charts easily, the functionality to view multiple graphs on one page is included. Users are able to plot other plots on different predefined spaces on the web page. When the plotting is loaded, only one graph will be visible. After clicking the 'Graph++'-button, a new graph space and button will be added. The new button will correspond with the new graph. Since each graph will have its own button, it is possible to overwrite the graph if it already has been used before this. Due to restrictions in the Dash library and

performance issues, the maximum number of graph spaces created is fixed. Therefore this feature is not dynamic but rather static. At this moment, the Visualizer has been capped with 10 separate figures, however, this can be changed in the source code if this is preferred.

5.3. Multi-file support

The data visualizer has support for different kinds of files and can visualize multiple files at once. Every file that is loaded should be either of the .csv, .xls, .txt or .tsv type. Users do not have to load files one by one. The application is designed so that it is possible to select multiple files and load them all at once. After a file is loaded, it converts into a Pandas data frame. All loaded data frames are visible in the ”Select files to project” drop-down menu. A data frame can be plotted by selecting it in the drop-down menu, setting all the parameters and clicking on which plot it should appear. It is also possible to select multiple data frames in the drop-down menu. Numerous data frames are merged into one data frame, which means that multiple data frames are treated as one new unique data frame (Figure 5.1). It is still possible to see every file independently. When multiple data frames merge into one, the ”Different Files” option will appear in the Color Label drop-down menu. When the Different Files option is selected, each different file will have its own color (Figure 5.2).

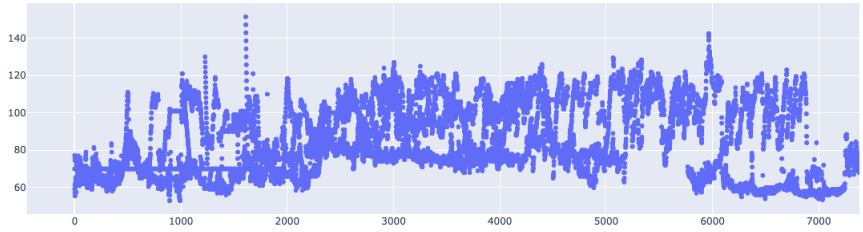


Figure 5.1: Multiple data frames which are merged into one data frame.

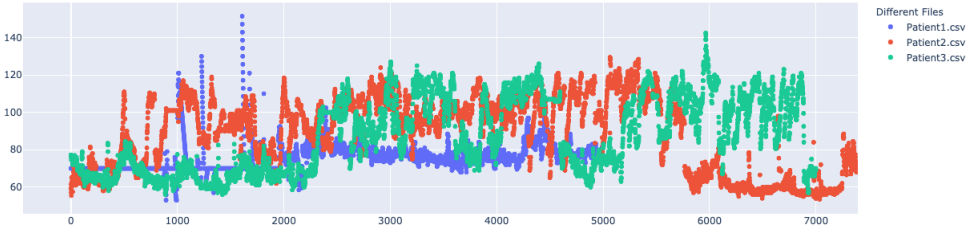


Figure 5.2: Multiple data frames which are able to be distinguished through different colours.

5.4. Nested Filtering

Nested filtering is a necessary functionality when examining data. At certain moments people only want to investigate parts of the data. This sometimes becomes cumbersome when loading in large amounts of data. Therefore it is useful to add a nested filtering option to exclude unnecessary data. Users do not have to filter their data beforehand but can load in raw data and filter it within the program. By clicking on the add-filter button, a new drop-down menu and text input area will appear. Users can choose on which column they would like the data to filter on and if the data must be (not) equal, larger, or smaller than a certain value. These values can be integers, floats and strings. It is also possible to filter on dates. When pressed on the apply-filter button, all added filters will be made to a single query. This query will be used on the data and return the filtered data. Users can also delete filters by pressing the delete filter, which is generated with every new filter. To reset to the original data, the user needs to press the reset-to-original-data button.

5.5. Data Selection

Another essential feature in the application is the possibility to select specific points in the graph. With these chosen points, the user can choose to display them in a separate graph. Additional filters can then be applied to this graph. The goal of this feature is to ease the process of finding correlations in

a dense graph. Due to dash restrictions, this task was not trivial. The selection from the graph was not returned; therefore and adequately a workaround was needed. The selected points in the graph are extracted from their respective data frame and subsequently highlighted in a table. In this table, the user can add or remove data points. At last, the user can choose to visualize this selection in a separate graph.

5.6. User-defined Functions

The functionality to write functions and implement this on the data was specifically requested by the client of this project. As a result, a separate directory called 'ClientCode' has been added. Here the user can add functions that are intertwined with the callback of a certain user-function button in the display. By default, two example buttons with two example functions have been added. When a button is pressed, the respective function is called. One important requirement for the User-defined functions is that the input and output should both be in the format of a Pandas data frame. With this functionality, the user can apply their own adjustments to the data without knowing how the back-end of the application functions.

5.7. Graphical User Interface Design

Good GUI design might save users about 40% decision-making time [11]. This is why we valued the design of our application very much. As can been seen in figure 1 the application contains a menu in a collapsible sidebar. This is purposely done to increase the potential size of the most important content of the application: visualized data. It also decreases the visual noise the user experiences while analyzing the graphs. The design is also very modular. Menus and buttons can be easily added or removed without tempering the functionality of the data visualizer. Modularity is of high importance since this will play a big role in extensibility. This modularity not only improves intuitiveness but also improves the maintainability of the application. This is because a developer does not have to search through huge files and structures to find the part which they want to improve. Instead, they only have to deal with small files that are precisely where a developer would expect to find them. This very modular design also contributes to the clients' wish of the application being extensible.

6

Project Assisting Tools

This chapter documents how we have used various openly available tools to assist us in creating the Interactive Data Visualizer. For every tool, it will be briefly stated what it is useful for, followed by a description of what we used the tool for and how we used it. In section 6.1, it is documented how GitLab was used, the most important tool for this project, for general planning and to keep an overview. In section 6.2 it is explained how and why the main libraries were used. And finally, in sections 6.3 PyBuilder and Selenium are described, which were the main tools used to test the application.

6.1. GitLab

GitLab is the most widely used tool for DevOps, which is a combination of software development and IT operations. We used a lot of the tools that GitLab has to offer, but the most important ones will be listed.

Firstly Git is one of the tools utilized. Git is by far the most widely used and supported version control system in the world. It is also Gitlab most important tool and is, as you might deduce from the name, what GitLab was built on top of. A version control system is very important when working on the same application with multiple developers. This is because developers typically work on different features or bugs that require different changes to the existing code. When a developer works on such a feature or bug, they temporarily work on a copied version of the application. Eventually, when multiple features are implemented or bugs are fixed. The developers obtained multiple different versions of the application. For instance, when two developers make a change in the same file, these changes might collide with each other. Then it must be decided how to merge the two versions back into one, which is done by using a version control system. Furthermore, Git is not only used to resolve these merge conflicts but also for general code review. In combination with GitLab, it allows developers to easily review each other's changes and comment on them before accepting them.

Secondly, an issue board was used, which was necessary to keep track of what everyone was working on and what still needed to be done. To do this, we used the issue board of GitLab. We used labels to differentiate issues based on priority, type, and status. For instance, when a developer started to work on an issue, they assigned themselves to it and changed its status to 'In Progress'. We also used the issue board for time management. When creating the issues, the necessary amount of time to be spent on them was estimated. And after a developer completed the issue and put their changes up for review, they indicated how much time they spent working on it.

Thirdly, a pipeline is used to ensure CI/CD. This pipeline uses PyBuilder as a runner a check after every commit whether the application still compiles and builds properly. Developers can easily review at which stage the pipeline failed or confirm that it succeeds after uploading their changes. GitLab also blocks attempted merge requests when the source branch failed the pipeline. This is to ensure that the target branch will always pass the pipeline and therefore have a functional version.

Furthermore, GitLab was used for various other things, such as having a general overview of the project, keeping the client up to date, and verifying whether different application versions are merged correctly. However, in the interest of conciseness, there will not be a further elaboration on these topics.

6.2. Main libraries

During the initial discussions, the client emphasized that the application should not be built from scratch. This is why a lot of libraries and frameworks were researched. The most suitable ones were picked and in this section, the motivation for picking these libraries and the way in which they were used will be described.

Plotly

The Plotly library is an interactive, open-source plotting library that supports over 40 unique chart types covering a wide range of use-cases [12]. Plotly is widely used and heavily documented online. It is also relatively easy to learn and the supporting website provides a lot of templates you can play with. It also provides interactive graphs that look great and that you could instantly publish if necessary. Furthermore, the Plotly library works very well with Dash, which is a framework made by the same company which we will explain in the next section. All of these reasons made it the best option for the Interactive Data Visualizer. The Plotly library was used to create all the necessary types of graphs. It was also used for zooming in and out and selecting and highlighting data points within those graphs. This was all possible thanks to the great interactivity of the Plotly graphs.

Dash

With Dash Enterprise, full-stack data science applications that used to require a team of front-end, back-end, and DevOps engineers can now be built, deployed, and hyper-scaled by a single data scientist [13]. Like Plotly, Dash is also very widely used and therefore has a lot of documentation online as well. It is also quite easy to learn and there are a lot of sample applications that developers can get inspiration from. The gallery of sample applications was used a lot since with the sample applications the actual code used to create them was also provided. This enables developers to not only follow tutorials and look at the documentation but also play with working applications. Dash was used for all of the front-end components such as drop-down menus and check-boxes. Ït was also used for all of the back-end processes such as callbacks, which are functions that are passed to other functions. The callbacks are used to communicate selected options and filters to interactively adapt the graphs. Furthermore, Dash was used to do all the styling and structuring of the application in the same programming language as the rest of the application, Python.

Dash OOP Components

To keep our code structured and have an overview of the functionality of each part of the program we used a lot of object-oriented programming concepts. The Dash OOP Component library enables developers to do this with Dash since it allows you to divide code into an object-oriented structure. This allows for composable, reusable, extendable, and fully configurable dash code.

Every class has a layout method where the dash components and HTML code are created. Aside from the layout, all classes also have a back-end part with the callback functionalities of the dash components. This results in a class where all the functionality and implementation are in one place. Before implementing the Dash OOP Components library, our code was relatively chaotic. All code was put in one python class, which made it unclear which method had what kind of functionality. Since our code is divided into well-structured classes, it is now a lot more readable. This also makes the application a lot more extendable since developers will know where to look when they want to improve the application.

Pandas

Pandas is a fast, powerful, and flexible Python package used to analyze and manipulate data. Since the objective was to build an application that visualizes data in a lot of different ways, a tool to handle this data was necessary. Pandas is used to read the data from the files that the user uploads when using the application. Pandas convert this data into a format that is usable by the rest of the application. Pandas is also used when the user wants to change the data by adding columns or when the application needs to merge two or more different data frames into one. Furthermore, using Pandas enables our application to use the same data frame for a whole variety of data visualization and manipulation operations. The same unconverted data frame can be used to plot the data using Plotly Express. However, it can also be used to manipulate the data by performing a statistical procedure on the data, such as

Principal Component Analysis, which is an algorithm that reduces a data frame to its most essential characteristics.

6.3. Testing tools

Every software developer will agree that testing your software is one of the most important steps of the development process. Furthermore, tests enable you to smoothly work on an application with a group of developers. This section will elaborate on this and on how tests were used to guarantee the functionality of the application.

PyBuilder

Since multiple people are needed to work on this application at the same time, some tools were required to make this a smooth process, such as PyBuilder. To understand why we decided to use this tool in particular, some other concepts must first be explained.

A CI/CD pipeline is a series of steps that are executed when a developer uploads the changes they made to the application. In a typical pipeline, there are three stages. In the first step, the application is built. In this step, the pipeline verifies whether the new version can be compiled. Secondly, all the available tests are run on the application, verifying that they all pass. Finally, in the third step, it is verified that enough code lines are tested.

All these steps require the application to be built automatically first. After researching the available tools, we decided to use PyBuilder since it is a build automation tool for the Python ecosystem. In the pipeline, PyBuilder is installed first, then a series of commands are run. These commands ensure that: the application is properly built, all tests are run, and verify a high enough test coverage.

Selenium

Selenium is used to automate the testing process for the data visualizer. Selenium is an open-source test framework that contains a suite of tools that is used for automating tests on browser applications. Selenium supports multiple browsers, so it is possible to verify that the application is working on several browsers [14].

For each Selenium command in the test script, an HTTP request is created and sent to the browser. On this browser, these commands are executed by the driver. These commands represent user actions that can be performed on a web page. For example, in the data visualizer, one could set a command in which a drop-down value is set. The resulting test from this could be to verify whether or not a graph pops up.

Using Selenium, it is possible to write use case tests using the coverage criteria for GUI testing. A. M. Memon, the head of the test strategy team of Apple Inc, states in [15] that testing must be done in a hierarchical way represented by an integration tree which is seen in Fig 6.1. The data visualizer can be tested from start to finish, simulating interactions between the user and the visualizer.

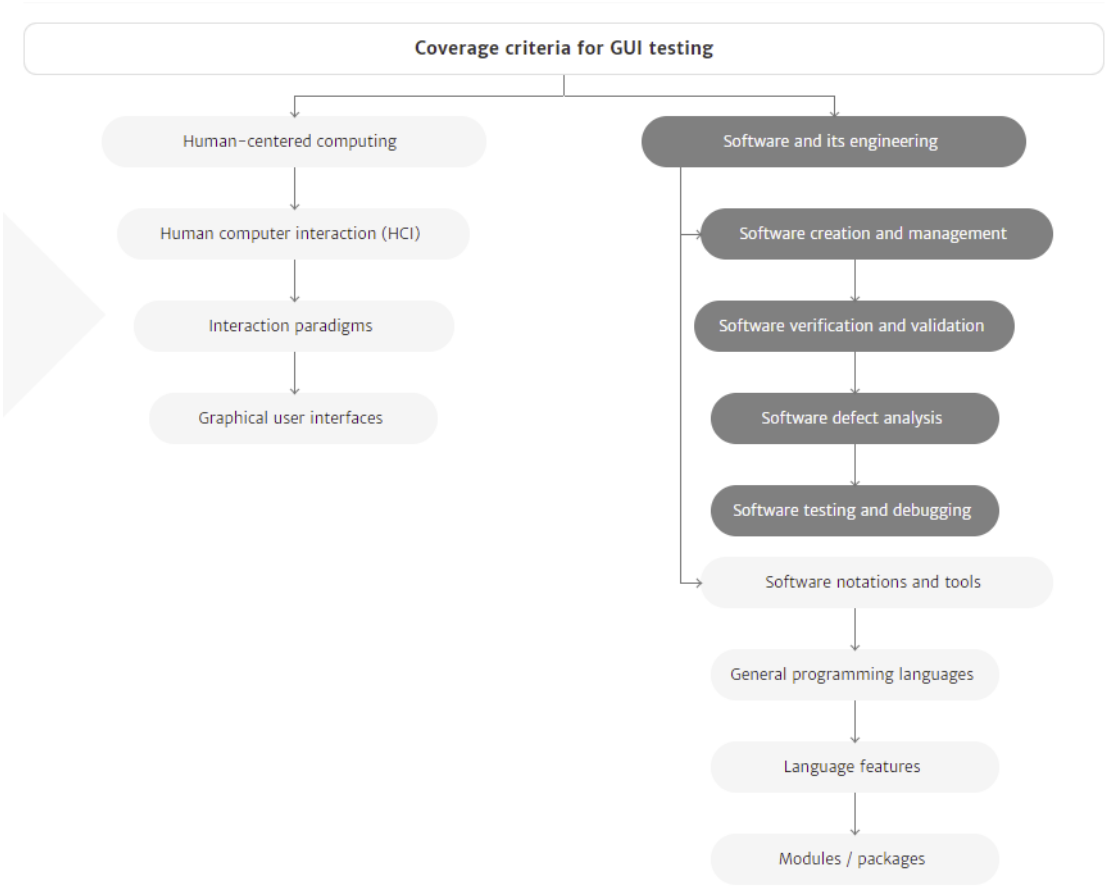


Figure 6.1: Integration tree for the coverage criteria of GUI testing[15].

7

Product and Project Evaluation

To build the app right or to build the right app? The answer to this question should of course be both! In section 7.1 we will delve into how we confirmed that we built the right app. In previous chapters, the tools that were used to confirm the application was functionally built right were already described. In section 7.2 however, the process of the implementation in terms of planning and teamwork will be elaborated on.

7.1. Application evaluation

Creating an application that merely suffices the requirements was never the end goal. Ultimately, the application was made to be used by actual people like our client. Therefore the team frequently met up with the client to get feedback on the application and suggestions for new features. However, the client was not the only stakeholder. The application's user experience was therefore also tested on a semi-random group of people.

Evaluation process

The people in the group had varying backgrounds to make sure that the application would be optimal for all sorts of users. Furthermore, the evaluation group had never seen the application before. This was to make sure that they were unbiased. While using the application they judged it based on a list of conditions such as functionality, clarity, intuitiveness, aesthetics, and uniqueness. How the application was initially judged can be seen in table 7.1. After considering this feedback the application was changed. It was then reevaluated to see if the changes lead to a better experience. Exactly how the changed application was evaluated and how it compared to the original application can be seen in table 7.2.

Condition x User (Field)	Damiaan (Mathematics)	Jerald (Mechanical Engineering)	Lucas (Industrial Design)	Thomas (Electrical Engineering)	Madeleine (High school)
Functionality	9.0	9.0	8.5	9.0	9.5
Clarity	7.0	7.5	6.5	6.5	7.5
Intuitiveness	5.5	6.0	6.0	6.5	6.0
Aesthetics	7.5	8.0	7.0	7.0	7.5
Uniqueness	8.5	9.0	8.5	8.5	9.5
Remarks	Upload button should be under ”File(s)”	It should be clearer that filtered data is only used for new graphs	Graph 1/2/3 options should be at the bottom of the menu	The main page should have some content	Why do they teach us to make graphs using excel?

Table 7.1: Table that shows how users from different fields rated the application based on a list of conditions.

Condition x User (Field)	Damiaan (Mathematics)	Jerald (Mechanical Engineering)	Lucas (Industrial Design)	Thomas (Electrical Engineering)	Madeleine (High school)
Functionality	9.0	9.5	9.0	9.0	9.5
Clarity	8.5	9.0	7.5	8.0	8.5
Intuitiveness	8.5	9.0	7.5	8.5	7.5
Aesthetics	8.0	8.0	8.0	8.5	7.5
Uniqueness	8.5	9.0	8.5	8.5	9.5
Remarks	-	-	-	-	-

Table 7.2: Table that shows the ratings and changes in ratings of how users from different fields rated the improved application based on a list of conditions.

Evaluation results

From the evaluation data, it is clear that the changed application performs a lot better. The average grades for all but one field have improved. The improvements for all the fields are: functionality +0.2, clarity +1.3, intuitiveness +2.2, and aesthetics +0.6. Only uniqueness has stayed the same. The changes had the greatest impact on intuitiveness. This was great since the average rating for this field was also the lowest. Therefore the main objective when adapting the application was also to improve in this field.

7.2. Planning and teamwork

Great results are rarely achieved with poor planning and teamwork. Therefore great importance was laid on these concepts. A lot of the planning that was done has already been described in the project plan that was made before implementation, which can be found in Appendix C. In this section, we will reflect on how well the initial planning was upheld and on how well the team members worked together.

Reflection on planning

The initial project schedule can be reviewed in Appendix B. Overall this planning was pretty well adhered to. However, some less important requirements were realized before some of the more important ones. In some cases, this was necessary since they relied on each other. In other cases, it was not necessary but was done out of convenience. For instance when a developer tries to implement an important feature and while doing this sees an opportunity to quickly implement a less important feature. When the planning was not adhered to, this was always because the team was ahead of schedule rather than behind.

In the most important tool, GitLab, the time tracking tools were also used pretty well. This allowed all team members to keep track of the time they and others estimated to spend and of how much they spent.

Reflection on teamwork

This reflection was, before any rewriting and correcting, written by one person. This is because within the team the opinions about how the team worked together were so aligned that the team trusted one person to write it. Before any implementation was done a code of conduct was written. In this document clear rules were made for timing, presence, productivity, deliverables, meetings, communication, and conflicts. We are confident that these elaborate rules caused the team to work together smoothly.

Another reason is the extensive usage of GitLab. The tools GitLab provides were created for reasons that were discovered during the implementation process. A lot of these tools were used by the team to aid progression of the project. It was because of this that it was always clear for someone what everyone else was working on and one could therefore always make a very well-informed choice of whom they should help or what new problem they should tackle.

During the implementation, some conflicts did arise, mostly for decisions about implementation or design. However, through civil discussion, a solution was always found. It was because of these reasons that everyone is satisfied with the process and very happy with the result. Everyone feels like they have developed themselves a lot, to become a better software engineer but also to become a better team player.

8

Product Discussion and Recommendations

This chapter provides an overview of the final product and recommendations for any future extensions. In section 8.1, an in-depth description is given on whether the end product meets the client's initial requirements. Section 8.2 elaborates on any recommendations we have for anyone who wants to further extend the application.

8.1. Product requirements

After confirming the right application was built in chapter 7 it will now be confirmed that the application was built right from a more exact standpoint. This section will discuss whether both the MoSCoW and the non-functional requirements are met.

MoSCoW requirements

Not all of the requirements listed in chapter 3 have been met. Fortunately, this only concerns two of the 'could have' requirements. The main reasons for not meeting these requirements are a combination of underestimated difficulty and limitations regarding the time frame given for the implementation. In the schedule we planned to start working on the 'could have' requirements by week 8 and to be done with them by the end of week 9. And although we started working on these requirements even earlier than week 8, we still did not allocate enough time towards these requirements.

The first one, the requirement of saving presets of selected menu options. During week 8 we discovered that exporting and importing configurations of the loaded data was harder than expected. Eventually, we did not have enough time to make this requirement fully working. While a different solution has been found in collaboration with the client, this solution is still not optimal.

The second requirement, which was to be able to run the application as a web app has also not been realized. The application does run in your browser, so porting it to a web application should not be very hard. However, we chose not to because the application could be used to visualize very sensitive data. And therefore rushing a step that heavily increased the chances of this data ever leaking felt like an irresponsible decision. To implement this properly more time should also have been allocated towards this requirement. Fortunately, all of the other requirements have been met.

Non-functional requirements

One of the most important non-functional requirements given by the client was that the application should be extensible. This is because the client plans to use the application for their own research in the future and this may require other functionalities. This is why the application's code needs to be well structured, easily interpretable, well documented and built on heavily documented frameworks. One of the most important decisions made to guarantee a clear structure was to separate our code in an object-oriented way. Furthermore, to make it easily interpretable the standards for well-written code were always adhered to. Pydoc documentation was also used to describe every function and there are a lot of in-line comments in place to further explain the logic. The README file also contains further

information on how the code is structured as well as how the code can be extended with for instance new functionalities or a new layout. Lastly, the whole application was built using a few frameworks. In choosing these frameworks it was always verified whether they were well documented. This was done not only to ensure that using them would go smoothly but also to ensure that anyone extending the application would not have a hard time understanding how to use them. A demo showing the actual code and its structure has also been given to the client to further enable them to extend the application in the future. Altogether, the client was very satisfied with the application's extensibility and stated that they will probably use it and extend it for their own data visualizations.

Besides the application's extensibility, there were other non-functional requirements. Most of these were satisfied, such as that the application should be built in Python and that it should be user-friendly to a certain extent. One requirement proved to be more difficult to satisfy. Which is that the application should be able to handle data set sizes of up to 5 gigabytes. In practice, the application's performance depends a lot on how much memory is dedicated to it. Furthermore, this issue might not be too relevant because a data scientist like the client will probably perform data manipulation methods that drastically reduce the size of their data sets anyway. Still, the application does allow these big data sets. However, it has a hard time visualizing them.

8.2. Future recommendations

Despite realizing the client's needs, the product could still be extended in multiple ways. In this section, the main reasons why we think the application is not optimal yet will be described. Furthermore, recommendations on how to tackle the issues related to these reasons are given for anyone who might try to extend the application in the future.

Saving and selecting configurations

One might want to review the current configuration of the application at a later time. One way they could do this is by just leaving the application running. However, this is arguably not optimal. Furthermore, someone might want to share their configuration of the application with colleagues to show them something interesting. They could do this by going to them or sharing their screen in an online call. However, this is also not optimal. A better solution exists and has already been researched. This is to be able to save, share, and load in previously made configurations. While this was not done successfully in the time frame given for the realization of this application, we do believe it is possible.

Porting to web application

At the moment, the application runs locally within the browser without any external connections. Since the application is built to run within a browser it could easily be extended to a web application. By making the Interactive Data Visualizer a web application, not only people that downloaded the application could use it. This would enable more people to use it and that might result in more valuable insights, which was the main reason for building this application in the first place. Anyone using the application could then also easily share it with their colleagues, which might set in motion a sort of exponential growth in the application's usage. The downsides of making it a web application have been described in chapter 4. While these downsides should definitely be considered we still recommend porting the application to a web application. However, we recommend that in this process cybersecurity experts are consulted.

Cross-filtering for graphs

A graph that is dependent on another graph could be useful for some use cases. For instance when you have a graph containing collective data of a lot of patients. One might want to inspect individual patient data in a second graph, for instance after hovering over a data point of the original collective graph. Although this feature was not a hard requirement the possibilities of it being realized were discussed with the client. The actual implementation of this feature is very dependent on how someone wants to use it. A sort of one-for-all solution is very hard to realize. Therefore it was decided to not implement this feature. Dash does support this functionality and there is a lot of documentation available on how to realize it. Therefore we recommend anyone who requires this feature to look into extending the application with their version.

The release of Dash 2

In the second half of 2021, Dash 2 will be released by Plotly. A lot of things are still unclear about it like the release date, the added functionality, and with that the relevance for this application. However, Dash was by far the most important framework for the implementation of the application. Therefore just knowing this might still prove quite helpful for anyone who is looking into extending the application.

9 Conclusion

The purpose of this report is to comprehend what features a data visualizer should be present for a common researcher. This was done by answering the following question: *How can we create a data visualizer with the current tools and software available online?* Different libraries and frameworks have been discussed to achieve this. Conclusively the choice had been made to primarily use the Dash library in combination with additional libraries. One of these additional libraries is the Dash OOP Components, which ensures extensibility for the data visualizer. The visualizer also heavily depends on the Pandas library, which is used to save the uploaded data into data frames. Furthermore Selenium is used to give the possibility to perform use-case testing.

These tools together form the answer to the centralized question in this report. By researching the already existing tools available online and combining them, most of the features can be realized. From nested filtering to selecting data in the graph. Combining multiple libraries makes realizing all these features possible. During the project, it came clear what features were missing in the application. This was discovered by requesting constant feedback from the client about the application. Furthermore, it has also come to our attention that the Dash library is not suited for dynamic applications. Adding graphs or buttons dynamically is not as trivial as expected and created a lot of overhead and creative thinking.

Several improvements are recommended. Firstly, as requested by the client, the application can be ported as a web app. This will make it possible to use the application online with other people consecutively. However, this will also raise ethical questions about data integrity and privacy concerns. Secondly, saving the workspace is a feature that did not make the deadline for the project. By saving the workspace all created graphs and their applied filters will be stored in a file. With this, the user can reuse their previous work and continue, as oppose to starting anew. Finally, displaying the correlations between graphs is a desirable feature. When multiple graphs are present in the application and their characteristics are equivalent it should be possible that their equivalent data points are highlighted.

Reference List

[1] J. Kiggins. Accessed Feb. 26, 2021. Hass Advocado Board. URL: https://www.kaggle.com/timmate/avocado-prices-2020.

[2] J. Bulao. *How Much Data Is Created Every Day in 2021?* Techjury, blog. May 2021. [Online]. URL: https://techjury.net/blog/how-much-data-is-created-every-day/#gref.

[3] T.H. Davenport and D.J. Patil. “Data Scientist: The Sexiest Job of the 21st Century”. In: *Harvard Business Review* (Oct. 2012. [Online]. url: https://hbr.org/2012/10/data-scientist-the-sexiest-job-of-the-21st-century).

[4] Z. Gemignani and C. Gemignani. *Juicebox*. Nashville, TN, USA, 2021. [Online]. URL: https://www.juiceanalytics.com/.

[5] U. Kulesza, V. Alves, A. Garcia, et al. “Improving Extensibility of Object-Oriented Frameworks with Aspect-Oriented Programming”. In: *Reuse of Off-the-Shelf Components*. Ed. by M. Morisio. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 231–245. ISBN: 978-3-540-34607-4.

[6] A.N. Jahfari. *Demo: t-SNE inspector*. Delft, The Netherlands, 2021. [Online]. URL: https://heartpr.shinyapps.io/tsne_pcas_removed/.

[7] S. Hatton. *Early prioritisation of goals book series, in Advances in Conceptual Modeling Foundations and Applications*. Vol. 4802,235- 244. 2007, pp. 517 –526.

[8] W. Maddox. *Why Medical Data is 50 Times More Valuable Than a Credit Card*. DMagazine, blog. Oct. 2019. [Online]. URL: https://www.dmagazine.com/healthcare-business/2019/10/why-medical-data-is-50-times-more-valuable-than-a-credit-card/.

[9] Daniel Strech and Neema Sofaer. “How to write a systematic review of reasons”. In: *Journal of Medical Ethics* 38.2 (2012), pp. 121–126. ISSN: 0306-6800. DOI: 10.1136/medethics-2011-100096. eprint: https://jme.bmj.com/content/38/2/121.full.pdf. URL: https://jme.bmj.com/content/38/2/121.

[10] Sung-Hwan Kim, Nam-Uk Kim, and Tai-Myoung Chung. “Attribute Relationship Evaluation Methodology for Big Data Security”. In: *2013 International Conference on IT Convergence and Security (ICITCS)*. 2013, pp. 1–4. DOI: 10.1109/ICITCS.2013.6717808.

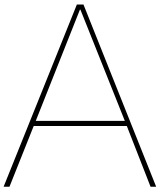
[11] W.O. Galitz. *The Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techniques*. 3rd ed. Indianapolis, IN, USA: John Wiley & Sons, 2007.

[12] Plotly. *“Plotly Python Open Source Graphing Library” Accessed June. 12, 2021*. URL: https://plotly.com/python/getting-started/.

[13] Plotly. *“Dash Enterprise is the low-code platform for ML & data science apps.” Accessed June. 12, 2021*. URL: https://plotly.com/dash/.

[14] P.V. Singh. *”What is Selenium? Getting started with Selenium Automation Testing” Accessed 16th june 2021*. URL: https://www.edureka.co/blog/what-is-selenium/.

[15] A.M. Memon, M.L. Soffa, and M.E. Pollack. *ACM SIGSOFT Software Engineering Notes*. Association for Computing Machinery, New York, NY, USA, 2001. [Online]. DOI: https://doi.org/10.1145/503271.503244.



Requirements Questions

These are the questions we asked our client for the requirements and the answers our client provided on 21-04-2021.

A.1. General

- **Question**: What do you mean by a top-down approach regarding the engineering process?

  Answer: *"Work in blocks, adding to the GUI piece-by-piece"*

- **Question**: On the project forum, we read software architecture is very important for this project, but what do mean by that exactly? What architectural patterns do you propose we use? (Client/server, MVC, Micro-services etc?)

  Answer: *"Pick the (useful) functionalities from the libraries. Make sure to build upon already existing functionalities. Furthermore the application should be extendable"*

- **Question**: Can workspace be provided?

  Answer: *"With current measures this will not be possible. The SP-staff however is looking for possibilities to arrange such needs in the near future"*

- **Question**: What are your working hours, to reach you for questions?

  Answer: *"Email is always possible at all times. Mattermost is preferred as most conversations will be back-and-forth"*

- **Question**: How often would you like to meet us?

  Answer: *"Once every 2 weeks, if necessary once a week; preferably Monday morning or after 14:00 || Friday whole day. These meetings can be arranged whenever needed through Mattermost"*

- **Question**: What mode of communication should we use to reach you?

  Answer: *"Mattermost/Email"*

A.2. GUI

- **Question**: Should we create our own design for the GUI, or do you have a design in mind?

  Answer: *"No particular preference. The given demo can be used as a guideline"*

- **Question**: What is the order of prioritization regarding GUI components (GUI, plotter, data selection, thumbnail, caching)?

  Answer:

  - *"Data selection (nested filtering): Including and excluding several variables and using that as a filter to narrow the scatter further (eg. Selecting a specific person in the plot and change a variable to years and select a specific year for that specific person).*
  - *Plotter: This will mainly be a scatterplot with an option to select a certain interval on top of the scatterplot. A possible option to smooth the scatterplot is possible*
  - *Thumbnail: Every point will contain meta-data and will create a small thumbnail displaying how the scatterplot will look like for that specific point (meta-data of specific point)*
  - *GUI: If possible, try to abstract a much a possible in such a way that the program can be expanded easily (API)"*

- **Question**: Do you have any tips on which GUI builder we should use?

  Answer: *"Ploty, Dash or Bokeh are good libraries to build upon"*

A.3. Typer data

- **Question**: Should the end product be able to convert data to a useful format itself, or will only converted data be provided (e.g. pictures to numbers etc.)?

  Answer: *"Data will always be delivered by us in the correct format. Coverting is not needed. Furthermore is you prefer the data to be in a (slightly) different format that is also possible. "*

- **Question**: Are we getting a specific data format?

  Answer: *"The data delivered will be time-dependant. It will be delivered in a table or CSV-like format. Metadata will be in a different table."*

- **Question**: What data set sizes can we expect?

  Answer: *"Data are quite big and in the size of several Gigabytes. It might be possible to make the user preselect certain variable to plot and preload the remaining variable in the background to save memory, increase efficiency and decrease overhead (lazy loading)"*

- **Question**: For visualization, how many dimensions will the data generally have? (How) should we handle (attempt to visualize) >3 dimensional data?

  Answer: *"Data will be preprocessed and converted in such a way that it will only have 3 or less dimensions."*

- **Question**: Besides graphs, should there also be an option to display the data in a table?

  Answer: *"Data will be visualized in a scatterplot. If possible continious variables could be plotted in a heatmap."*

A.4. Client/server division

- **Question**: Data only local or should it be taken from a server? Or Both?

  Answer: *"The program and the data will be local. In case of expansion to a WebApp is will be online. However, if that happens necessary administrative measures must be taken to preserve the anonimity and security of the data"*

A.5. Non-functional

- **Question**: How user-friendly should the GUI be? Accessible for people without ML background?

  Answer: *"The only user that will be using the program will have a background in machine learing"*

- **Question**: Should there be an explanation of what the tools do?

  Answer: *"Documentation of how the program work back-end as well as front-end is recommended"*

- **Question**: Should users be able to have an account?

  Answer: *"Such thing is not necessary"*

- **Question**: Should users be able to save presets of settings (ticked boxes etc.)?

  Answer: *"An expansion that would be nice to have*

- **Question**: In what format must the final product be in? (eg. .exe, .py2exe)

  Answer: *"The final deliverable should be an executable runnable in Linux"*

A.6. Webapp (extra)

- **Question**: How important is a web app? Should we prioritize it over some of the other requirements or treat it as the last requirement?

  Answer: *"This should be the biggest priority after the main program is finished. A very big wish, but not a requirement"*

A.7. Machine learning

- **Question**: What are the minimum amount of tools we should have in our visualiser?

  Answer: *"The use of numPy and sciPy is definetely necessary since most of the data is also generated with datastructeres from these libraries"*

B

Schedule

	Week#	1	2	3	4	5	6	7	8	9	10	11
	Monday	19-4-2021	26-4-2021	3-5-2021	10-5-2021	17-5-2021	24-5-2021	31-5-2021	7-6-2021	14-6-2021	21-6-2021	28-6-2021
Planning	Research											
	Interview Stakeholder											
	Project Plan (incl. Solution Proposal)											
Design	GUI Mock-Ups											
	Architecture											
Implementation	Set-Up Dash Application											
	Implement Graphing											
	Implement Data Filtering											
	Add visuals to GUI											
	Implement Must Have											
	Implement Should Have											
	Implement Could Have											
	Extend to WebApp (optional)											
	Test Code											
Reflection	Meetings w/ Client											
	Meetings w/ TA											
	Meatings w/ Coach											
Documentation	Final Report									A10 17-6		
	TW Assignments	A1 23-4	A2 28-4	A3&4 6-5	A5 12-5	A6 20-5		A7 1-6	A8 9-6 & A9 13-6			
	Final Presentation									Prep	Prep	

C

Project Plan

Project Plan

Interactive Data Visualization

Glenn van den Belt
Shaan Hossain
Joost Jansen
Adrian Kuiper
Philip Tempelman

TU Delft

Contents

1 Problem Analysis

1.1. Main goal

As datasets grow larger and larger, performing computations with them becomes cumbersome and slow. In addition to this, deriving valuable information from these immense datasets is not always trivial. Our goal is to create an efficient and extensible application that visualizes data into useful graphs. This visualizer should be interactive and when using it the user should be able to derive valuable information from the data, for instance by recognizing possible correlations between different data features. Furthermore, the user should be able to use the features to filter out irrelevant or unwanted data. Finally, the client has emphasized that building such an interactive visualizer should not be done from scratch unless doing so would otherwise be infeasible.

1.2. Libraries

Picking a suitable library that meets our needs is a primitive task. After discussing, we have concluded that to meet our client needs, we need an extensive, high-level library and fully use its functionalities or use a low-level library and build all the functionalities from scratch. Most high-level libraries are not extendable in terms of features and are thus limited to what we can do with them. The other possibility is to use low-level libraries and build all the functionalities from scratch. Low-level libraries give us more freedom but will be more complicated since we will have to create everything from the ground up and combine multiple different libraries.

1.3. Type of data

The data will always be formatted in such a way that it fits the application best. This data will have a time series format and might sometimes contain continuous variables. In case of continuous variable the visualizer must change into a heat map instead of a scatter-plot. Furthermore, each data points must each carry its own meta-data and must be visualized when hovered over them. This metadata can be either delivered in a table separate from the actual data and linked with relational properties.

1.4. Nested Filtering

All data points in a scatter plot are independent, and each carries its own meta-data. According to the user needs, additional filters could be applied when viewing a subset of data points. This phenomenon is called nested filtering, as multiple filters may be applied to a set of data points. When using multiple filters, we must avoid parsing the whole dataset again due to efficiency and memory reasons. We must find a convenient way to store intermediate results while minimizing memory use, such as caching.

1.5. Functions and Algorithms

After the dataset has been plotted the user will need to be able select a set of data points. The user can give implement their own function or algorithm (t-SNE, PCA, etc.) and this will be applied over the selected data points. The result of this will be outputted and plotted. This function or algorithm will be

written by the user themselves in the back-end. We will have to ensure that the plotter will work with any given well-defined function.

1.6. Computation intensity and Thumbnails

As the client provides us with a large amount of data (in the order of Gigabytes), it might be cumbersome to load all the data in at once. The client has pointed out that a feature should be created, which allows the user to give an order of which part of the data is loaded first. We should keep this feature in mind when expanding the project to a web application, since the browser may be limited in memory and could raise some problems due to efficiency reasons. When for example, a filter box is ticked, the application will query the whole database, and therefore the loading time of the application will take a lot longer. Additionally, to be able to filter on specific points, the application also needs to create a small thumbnail that visualizes a plot of that specific point when hovered over it.

1.7. Extensibility

The client has indicated that they should be able to extend the visualiser with additional charts, graphs, filters and/or similar things. Therefore we have to create some sort of framework which makes it straightforward for our client to extend the visualiser with additional features in the future.

2

Requirements Elicitation

In this chapter we will discuss the requirements deduced from the information we retrieved from the client. We will explain both the functional and non-functional requirements using the MoSCoW method.

2.1. Process

Before meeting with our client, we thought about the stakeholders involved. Since the only stakeholder involved is our client, this made the requirement elicitation process relatively simple. In preparation for the meeting, we have constructed a set of questions. These questions are based on the project description, which is found on the TU Delft Project Forum. Our questions were constructed in such a way that they can find the needs of our client effectively.

During the meeting with our client we have noted the answers on our questions. These questions and answers are listed in Appendix A. Besides answers to our questions, an oversimplified demo application [1] made by the client was provided.

2.2. Findings

The client indicated that they are working with very rich time-series data extracted from heart rhythms disturbance patients, such as information on the step counting, heart rate bpm, and more. In addition, our client also indicated that handling this data is often a pretty complicated task and that ”just putting everything into an algorithm” is usually quite useless. Therefore, our client would like an application that visualizes the patient data in an interactive way in which he is able to retrieve valuable information.

The dataset size of patients which needs to be handled are often extending multiple gigabytes in size. To efficiently manage data of this size, the client has suggested that our application should load the data lazily. In other words, when only the data of one particular patient needs to be plotted, the application doesn’t have to load the data of all patients. Lazy loading will make the application more time efficient, especially when only subsets of the entire dataset have to be plotted.

Furthermore, the client demanded a feature that enables users to perform nested filtering. For example, when plotting the data of all patients, a user should be able to select a particular patient and plot the data from only the year 2010 for that specific patient.

Our application needs to make use of methods like t-distributed stochastic neighbour embedding and principal component analysis to reduce our rich data into plottable data.

2.3. MoSCoW Method

When successfully forming a general description of the end product, our client demanded a more specific approach to build the application. To apply for the client needs, we have formed a list of requirements using the MoSCoW method. A list of these requirements is visible in Appendix B.

3

Feasibility Study

Subsequently to making requirements with our client, we've started to raise the question of the feasibility of the project. We've looked at the current conditions, technicality and the time constraint of the project and came to the following conclusions.

3.1. Technical Feasibility

Looking at the project's technical requirements and the technology currently available, we believe that it is possible to succeed in the project since our project is similar to multiple products online. Take, for example, the Support Vector Machine (SVM) Explorer listed on Dash Enterprise App Gallery.[2] This product displays data in a graph with multiple parameters to adapt the interpretation of the SVM. The difference between this product and our project is that our project will have various tools and not only SVM. The extension of our project compared to the already existing projects should be technically feasible with existing libraries.

3.2. Operational Feasibility

Our client will use our project for visualizing patient data. With the current requirements listed, this should be possible with our end product. Aside from the current requirements, our product must be extendable. Therefore, if our client wishes to change its need for the product, the product itself can be adapted. Thus we believe if our project will be developed and implemented, it will be used for its need and thereby making our project operationally feasible.

3.3. Schedule Feasibility

Looking at the project schedule in Appendix C, we believe we can stick to the planning we made. As the deadlines for the Technical Writing assignments are very clear and the issues are evenly distributed and prioritized according to the MoSCoW method over the weeks, there should not be any problems finishing the tasks within the required time. Moreover, as no team member has any other courses to follow next to the software project, the focus will solely lie on working on the project. However, if we cannot complete the tasks before our deadline, then there is still room to change the schedule a little. We believe that a bit of flexibility is an excellent property to have. The program provided in Appendix C is largely feasible since each team members have 40 work hours available to work on the project. We think that this is enough time to deliver at the least the minimum viable product, but we are determined to provide more than that.

3.4. Summary Feasibility

To conclude our feasibility study, we think our project is technical, operational and schedule feasible given the current requirements.

4

Risk Analysis

Although our project does not deal with a lot of stakeholders and is not dependent on a lot of people or things there is still some risk involved, like with any project. In this chapter, we will briefly describe the biggest risk factors and how we will be dealing with them.

4.1. Sensitive data

4.1.1. Risk

As stated in chapter 2 we are dealing with very big datasets on heart rhythm disturbance patients. The data contains not only aggregated data points of a lot of patients but also individual data of each patient over time. Whether aggregated or not, this data is obviously very sensitive. Of course, a leak of any data is not desirable, but especially a leak exposing medical data would be terrible. If this data is not handled carefully not only the patients involved will be in trouble, but everyone responsible for the data might experience some harsh measures if it can be showed that they didn't do everything in their power to handle the data responsibly.

4.1.2. Solution

To prevent such a leak and all the following measures involved from happening we will (initially) make the application run locally, on local data. This way our application does not need to send or receive any data to and from online resources. Since by far most hacks and leaks occur over an internet connection and our application doesn't need any internet connection we greatly reduce the risk of the data being compromised. Eventually, we might turn the application into a web application, which was a could-have requirement from the client. If we choose to do so a lot of time and effort should be spent on making sure our application is impenetrable before we decide to deliver it to the client. Finally, we should also try to get some consulting by an expert in the data security field and adjust the application accordingly.

4.2. Lack of knowledge of libraries

4.2.1. Risk

The usage of the correct library is determinant for our whole project. This library will be the base of our project. Most requirements will only be considered after the base layer of the application has been implemented. There is a possibility that we only then find out that the library we have used is very limited in terms of adjustments and extendability. This would result in starting the project over with a different library or not to use a library at all and create one ourselves.

4.2.2. Solution

To prevent choosing a library that does not meet our needs we have done some experimenting with them. After some experimenting with several libraries and arbitrary datasets we have gotten a modest idea of the capabilities of these libraries. We studies the documentation of these libraries and found several articles and videos explaining and demonstrating the possible use cases of these libraries.

5

Project Approach

5.1. Use of libraries and frameworks

We have done meetings with our client, the teaching assistant and the coach. During these meetings, we've got suggestions on what tools to use in building our final product. After doing additional research, we've found more tools. We tried to understand all the tools and compared them with each other. In the following sections, we will explain which tools we've found, what they are helpful for, and eventually, we will motivate our final decision on what tools to use.

- **Scikit-learn**: *"a simple and efficient tool for predictive data analysis".*[3] Scikit-learn only predicts data using machine learning algorithms and doesn't project it. Since the main focus of the project is to visualize the data, we won't use it.
- **Kivy**: *"a cross-platform Python framework for NUI Development".*[4] The main focus of Kivy lies in GUI development. In our framework, we would like to visualize data. Since the focus of this framework lies in GUI development and not in data visualization, we can't use it solely, however we could use it to implement our GUI.
- **Bokeh**: *"a Python library for creating interactive visualizations for modern web browsers".*[5] This is close to what we are looking for. However, the focus of Bokeh lies in optimization for web browsers, and since we aren't initially building a web app, this is not exactly what we need. Bokeh might become a perfect option once we are willing to expand the project and port it as a web application in the future.
- **Matplotlib**: *"a low-level library which offers a lot of freedom at the cost of having to write more code".*[6] Matplotlib is very good for creating elementary graphs. If you would like to generate more complicated diagrams, such as 3D graphs, you need to write a lot of extra code to implement it. Our client has strongly suggested that the goal of this project is not to implement everything ourselves from scratch but rather to use already existing libraries. Matplotlib could work for our project since we could try to implement everything ourselves. However, this might not be feasible due to time constraints. Thus, we won't use Matplotlib primarily since it is not a high-level library.
- **Pandas**: A library for encapsulating and modifying data. This library will become helpful when the data is embedded in a specific format and needs to be reformatted to fit the application's input.
- **Plotly**: A framework that creates graph and table from a given dataset. This library is low-level respective to its competitors. Since Plotly is hosted on a server and our client would like that his data is only used locally, this might not be the ideal library we are looking for.
- **Dash**: *"A python user interface library for creating analytical web applications".*[7] Think about data visualization exploration and analysis. Dash can easily integrate with Plotly, a framework that makes a graph and table from a given dataset. Dash can build an interface to show the visualizations made by Plotly. Since Dash and Plotly can be integrated, it is an excellent option to use when we want to be able to expand the project and port it as a web application. However,

a downside could be that there is less room for potential new features since Dash is a high-level library.

We think that Dash is the best library to pick as a basis for our project. If Dash doesn't provide enough room for new features, we could try to introduce a more low-level library in addition to Dash, for example, Matplotlib. However, introducing another library is not favoured since it would mean we have to integrate multiple libraries, which can be challenging. To implement the GUI we consider using Kivy. According to our client, expanding the project to a web app is a huge nice to have. To do this, we could use Bokeh. Yet, this would lead to another integration with libraries, and therefore we prefer to use Dash to expand to a potential web app.

5.2. Efficiency

The application should be able to handle large datasets (of multiple gigabytes). Therefore efficiency is very critical. Luckily the Dash library provides various options for improving performance. Suppose the application ends up not running smoothly on the given datasets. In that case, we can use memoization, which is relatively easy to implement since the Dash callbacks are functional (they don't contain a state) in nature [8]. We can implement this by using, for instance, the Flask-caching library [9] to save and reuse the results of the Dash callback functions.

"First, it is often useful to take a quick look at the contents of the file before loading. It can be beneficial to see how a data-set is structured beforehand and what parameters we need to set to load in the file".[10] To prevent unnecessary loading, we will give the user the option to choose the perimeters they want to plot. Moreover, for the initial loading of large datasets, the application will load the whole set into chunks. The size of such a chunk will be dependent on the total size of the dataset. The chunks will then be loaded into the application

5.3. Quality Assurance

During the project, we'll use several methods to ensure our quality of the project. Each technique and to what extent it helps us improve our work's quality is described below.

5.3.1. Scrum

A crucial part of software development is choosing the proper software development methodology. These exist to guide the project through the development process to make this go as smooth as possible and create little ambiguity. Since we aim to keep the client engagement high for constant feedback and potential adjustments, scrum is also highly applicable.

As we are in week 4.2 and the deadline of the final product is at the end of week 4.9, we have chosen to keep our sprints one week long. Before starting the sprint, we will set up a plan of what must be done by the end of the sprint. At the end of the week, we will reflect on what we accomplished so far and how we can improve on it.

Daily scrum meetings will not be necessary as we aim to work together as much as possible. Whenever someone request aid in his work, it will be provided at almost immediate times. During the weekly meeting with the TA, we will present our progress. We aim to have meetings with our client every two weeks to show progress and offer both the client and development team opportunities to ask questions and offer critique.

At the end of every sprint, a sprint review will take place to document what we have accomplished so far, and problems will need to be tackled first in the upcoming sprint. Every two weeks, we will have this meeting with our client, so we can collectively decide on improvements and changes. If necessary, this meeting with the client will be held more frequently. During the retrospective, the team will discuss what went well during the sprint and how we can improve the development process.

5.3.2. Documentation

In the documentation, we'll describe every development of the project. When we run into problems, we'll document the problem and the solution which we've found in the final project. Examples of such issues could be that a library isn't fulfilling our needs and that we have to use another library or code a

solution from scratch ourselves. We also want to use proper documentation in our git repository. Every time a git action is performed, the documentation should try to follow these guidelines [11]:

- Give a short summary (50 chars or less)
- More detailed explanatory text can be provided, if necessary, in the body of the git command.
- Write the commit messages in the imperative. For example, write ”Fix bug” and not ”Fixed bug” or ”Fixes bug”.
- Add a reference to the issue at the issue board.

Lastly, there should be proper documentation of our Python code. We cannot assume all our code is self-explanatory, and therefore documentation in code is essential. To achieve this, we could use documentation generators such as sphinx or pdoc.

5.3.3. Software Testing

Since our program we'll be a straightforward GUI, we'll only use structural testing techniques. Steve Cornett stated that *”Code coverage of 70-80% is a reasonable goal for system test of most projects with most coverage metrics. Use a higher goal for projects specifically organized for high testability or that have high failure costs. Minimum code coverage for unit testing can be 10-20% higher than for system testing.”* [12]. Our program does not have high failure costs, but a well-working program is favourable. Therefore we'll set a minimum of 85% path coverage. Aside from code coverage, we'll be testing our project with Unit, integration and system testing. For this, we will use PyTest and PyUnit.

5.3.4. Version Control

To ensure version control, we are using git. For every new feature, we create a new branch. In addition, for every new piece of code or bug fix, we should make a commit on git. When implementations in a branch are done, we should merge the branch with the developer branch, which contains the up till now final product. The developer branch will be merged with the master branch if and only if the client agrees with the developer branch. In case the client is not satisfied with the contents of the developer branch, we can roll back the developer branch or change the content to the clients' needs.

5.3.5. Static Analysis Tools

To check our code statically, we will make use of a Pylint, a python style checker. Pylint is equivalent to checkstyle as it checks for coding convention violations. Conveniently Pylint also has error detection, which will detect whether declared interfaces are actually implemented and if added modules are actually used. These tools combined with testing can all be seamlessly run in a CI/CD pipeline in GitLab.

5.3.6. Definition of Done

A requirement is considered completed if and only if the client agrees on its functionalities. The project is considered done when all 'Must Have' and 'Should Have' requirements are completed. We do, however, aim to complete the 'Could Have' requirements as well. But since these are optional, we will not include them in our definition of done.

6

Solution Proposal

In this chapter, we will first describe how we propose to build a product that satisfies the earlier mentioned won't-, must- and should-have requirements. Afterwards, we will also explain how we can extend our product to satisfy some of the could-have requirements.

6.1. Base solution

This section includes a general description of the application that satisfies the won't-, must- and should-have requirements.

6.1.1. General

Our application will be a GUI written in Python. It will not have a database, but it should be able to read correctly formatted .csv files. The application will not do the formatting/data conversion itself. The data will be delivered in the desired format by the client. The application will be built using functionalities from existing libraries, and most functionalities should not be built from scratch.

6.1.2. Libraries

Most, if not all, of the requirements, should be able to be satisfied efficiently using the Dash library from Plotly. This library is perfect for interactive data visualization and is used by over 50M users. Some requirements may require more low-level functionalities. To satisfy these requirements, we will use functionalities from libraries like Matplotlib and Pandas.

6.1.3. Data manipulation

Since we need to visualize very rich (multi-dimensional) data, we need to reduce this data to 2 or 3 dimensions. For this, two options will be available to the user: t-distributed stochastic neighbour embedding and principal component analysis.

6.1.4. Data format

The data will be delivered by the client and will be in a .csv format. Furthermore, the data will be formatted and converted by the client to suit the application instead of the other way around.

6.2. Extensions

This section includes a general description of the extensions that need to be built for our application to satisfy the could-have requirements.

6.2.1. Web App

Eventually, the client would like the application to be able to be used by others. By people from their research department as well as by their colleagues from all over the world. To accomplish this most efficiently, the application could be a web app. This means that the application should be hosted on a web server and should ideally always be accessible.

6.2.2. Alternative visualization

The application could eventually be extended to visualize more sophisticated machine learning methods. Besides the standard scatter plot, we may include alternative approaches that could benefit from being visualized, such as classification methods like k-means clustering and support vector machine.

7

General Planning

7.1. Processes

Scrum: We will be using the Scrum methodology during the Software Project, as we will be having weekly meetings, where we discuss what the team is going to do for the upcoming sprint by prioritizing the issues that should be started on and how much time it will take to complete/solve this issues. Throughout the week, the team members will work on their assigned issues and document what they have done and how much time their task has taken. At the end of the week, the team will give feedback to each other on the tasks done in the previous week and whether it is completed sufficiently or if there is still room for improvement.

7.2. Tools

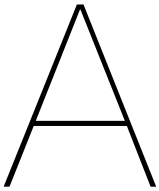
GitLab: For the project, we will collaborate by using GitLab. GitLab is also useful for keeping the issue board up to date. We will go over the issue board during every weekly meeting and assign the issues as a task to work on for the upcoming sprint. The issues may consist of bugs, new features, and tasks, whereas a description for every issue is given to guide the team to approach this issue.

Mattermost: During the project, we will be using Mattermost to communicate daily with our TA and our client. Thereby having a quick informal correspondence to ask small or big questions to our TA or client which will result in an fast adaptive decision making. Aside from communication, we'll use Mattermost to keep up-to-date with the latest announcements from Brightspace.

Discord: All of our daily meetings are held online on Discord. Discord is a productive tool for us because we use all of its main features. We use sub chats to ask questions on different topics and use screen sharing to solve problems we usually solved looking at the same screen. During our work hours, we're constantly on a call to have a quick communication about the topics we're currently working on.

7.3. Schedule

Our schedule of the project can be viewed in Appendix C.



Requirements Questions

These are the questions we asked our client for the requirements and the answers our client provided on 21-04-2021.

A.1. General

- **Question**: What do you mean by a top-down approach regarding the engineering process?

  Answer: *"Work in blocks, adding to the GUI piece-by-piece"*

- **Question**: On the project forum, we read software architecture is very important for this project, but what do mean by that exactly? What architectural patterns do you propose we use? (Client/server, MVC, Micro-services etc?)

  Answer: *"Pick the (useful) functionalities from the libraries. Make sure to build upon already existing functionalities. Furthermore the application should be extendable"*

- **Question**: Can workspace be provided?

  Answer: *"With current measures this will not be possible. The SP-staff however is looking for possibilities to arrange such needs in the near future"*

- **Question**: What are your working hours, to reach you for questions?

  Answer: *"Email is always possible at all times. Mattermost is preferred as most conversations will be back-and-forth"*

- **Question**: How often would you like to meet us?

  Answer: *"Once every 2 weeks, if necessary once a week; preferably Monday morning or after 14:00 || Friday whole day. These meetings can be arranged whenever needed through Mattermost"*

- **Question**: What mode of communication should we use to reach you?

  Answer: *"Mattermost/Email"*

A.2. GUI

- **Question**: Should we create our own design for the GUI, or do you have a design in mind?

  Answer: *"No particular preference. The given demo can be used as a guideline"*

- **Question**: What is the order of prioritization regarding GUI components (GUI, plotter, data selection, thumbnail, caching)?

  Answer:

  - *"Data selection (nested filtering): Including and excluding several variables and using that as a filter to narrow the scatter further (eg. Selecting a specific person in the plot and change a variable to years and select a specific year for that specific person).*
  - *Plotter: This will mainly be a scatterplot with an option to select a certain interval on top of the scatterplot. A possible option to smooth the scatterplot is possible*
  - *Thumbnail: Every point will contain meta-data and will create a small thumbnail displaying how the scatterplot will look like for that specific point (meta-data of specific point)*
  - *GUI: If possible, try to abstract a much a possible in such a way that the program can be expanded easily (API)"*

- **Question**: Do you have any tips on which GUI builder we should use?

  Answer: *"Ploty, Dash or Bokeh are good libraries to build upon"*

A.3. Typer data

- **Question**: Should the end product be able to convert data to a useful format itself, or will only converted data be provided (e.g. pictures to numbers etc.)?

  Answer: *"Data will always be delivered by us in the correct format. Coverting is not needed. Furthermore is you prefer the data to be in a (slightly) different format that is also possible. "*

- **Question**: Are we getting a specific data format?

  Answer: *"The data delivered will be time-dependant. It will be delivered in a table or CSV-like format. Metadata will be in a different table."*

- **Question**: What data set sizes can we expect?

  Answer: *"Data are quite big and in the size of several Gigabytes. It might be possible to make the user preselect certain variable to plot and preload the remaining variable in the background to save memory, increase efficiency and decrease overhead (lazy loading)"*

- **Question**: For visualization, how many dimensions will the data generally have? (How) should we handle (attempt to visualize) >3 dimensional data?

  Answer: *"Data will be preprocessed and converted in such a way that it will only have 3 or less dimensions."*

- **Question**: Besides graphs, should there also be an option to display the data in a table?

  Answer: *"Data will be visualized in a scatterplot. If possible continious variables could be plotted in a heatmap."*

A.4. Client/server division

- **Question**: Data only local or should it be taken from a server? Or Both?

  Answer: *"The program and the data will be local. In case of expansion to a WebApp is will be online. However, if that happens necessary administrative measures must be taken to preserve the anonimity and security of the data"*

A.5. Non-functional

- **Question**: How user-friendly should the GUI be? Accessible for people without ML background?

  Answer: *"The only user that will be using the program will have a background in machine learing"*

- **Question**: Should there be an explanation of what the tools do?

  Answer: *"Documentation of how the program work back-end as well as front-end is recommended"*

- **Question**: Should users be able to have an account?

  Answer: *"Such thing is not necessary"*

- **Question**: Should users be able to save presets of settings (ticked boxes etc.)?

  Answer: *"An expansion that would be nice to have*

- **Question**: In what format must the final product be in? (eg. .exe, .py2exe)

  Answer: *"The final deliverable should be an executable runnable in Linux"*

A.6. Webapp (extra)

- **Question**: How important is a web app? Should we prioritize it over some of the other requirements or treat it as the last requirement?

  Answer: *"This should be the biggest priority after the main program is finished. A very big wish, but not a requirement"*

A.7. Machine learning

- **Question**: What are the minimum amount of tools we should have in our visualiser?

  Answer: *"The use of numPy and sciPy is definetely necessary since most of the data is also generated with datastructeres from these libraries"*

B

Requirements

This appendix includes a list of the requirements ordered using the MoSCoW model.

B.1. Non-functional

- The system will be able to select, filter and visualize data within 2 minutes.
- The system should be user-friendly to the extend that people from all machine learning backgrounds are capable of understanding how to use it within 15 minutes.
- The system must be able to handle dataset sizes of up to at least 5 gigabytes.

B.2. Must Have

- The system must be built in Python 3.
- The system must be able to run in Linux.
- The system must have an easy to use graphical user interface.
- The system must be able to select data using an selection method such as 'rectangle selection' or 'lasso selection' or something similar.
- The system must be able to plot data points with a scatter plot.
- The system must be able to use multiple machine learning tools written by the user on the loaded data.
- The system must be able to show the meta-data of a specific point in a thumbnail, which for example pops up when hovering over that specific data point.
- The system must be able to visualise data which has three or less dimensions.
- The system must be able to run locally.
- The system must be able to run on data written in a csv file.
- The user must be able to filter on specific points.

B.3. Should Have

- The system should be extendable so that the client can implement additional data visualization and filtering methods.
- The system should have a short description in which the current tool of visualizing the data is explained.

- The system should be able to plot continuous data with a heat map.
- The system should support to inject any functions before plotting. An example would be a function to compute a density map before plotting or plotting the trajectory of data over time by connecting data points with lines and arrows.
- The user should be able to give an order of loading when a dataset is loaded.
- The user should be able to export and download figures made by the system.
- The system should avoid parsing the whole dataset again when applying multiple filters.
- The system should support nested filtering.
- The system should be able to create multiple subplots from one large plot and view them in on single time series

B.4. Could Have

- The system could run as a web app.
- The user could save presets of settings, in the form of ticked boxes etc.
- The user could be able to load in a csv file and select, filter and visualize the data.
- The system could be able to load in a csv file with meta-data within the desired table or with meta-data in a separate table.
- The user could use a Drag Drop to upload the data.
- Selection of specific colors for specific datasets.

B.5. Won’t Have

- Ability of users to have an account.
- Converter of data, since the client will deliver data in our demanded format.
- There won’t be any support for any local/online databases, since all data will be loaded in
- There won’t be support for mobile and tablet versions.

C

Schedule

	Week#	1	2	3	4	5	6	7	8	9	10	11
	Monday	19-4-2021	26-4-2021	3-5-2021	10-5-2021	17-5-2021	24-5-2021	31-5-2021	7-6-2021	14-6-2021	21-6-2021	28-6-2021
Planning	Research											
	Interview Stakeholder											
	Project Plan (incl. Solution Proposal)											
Design	GUI Mock-Ups											
	Architecture											
Implementation	Set-Up Dash Application											
	Implement Graphing											
	Implement Data Filtering											
	Add visuals to GUI											
	Implement Must Have											
	Implement Should Have											
	Implement Could Have											
	Extend to WebApp (optional)											
	Test Code											
Reflection	Meetings w/ Client											
	Meetings w/ TA											
	Meatings w/ Coach											
Documentation	Final Report									A10 17-6		
	TW Assignments	A1 23-4	A2 28-4	A3&4 6-5	A5 12-5	A6 20-5		A7 1-6	A8 9-6 & A9 13-6			
	Final Presentation									Prep	Prep	

Reference List

[1] Arman Naseri Jahfari. *Demo: t-SNE inspector*. URL: https://heartpr.shinyapps.io/tsne_pcas_removed/.

[2] Plotly. *Support Vector Machine (SVM) Explorer*. URL: https://dash-gallery.plotly.host/dash-svm/.

[3] Scikit-learn. *Simple and efficient tools for predictive data analysis*. URL: https://scikit-learn.org/stable/.

[4] Kivy. *Kivy - Open source Python library for rapid development of applications that make use of innovative user interfaces, such as multi-touch apps.* URL: https://kivy.org/#home.

[5] Bokeh. *The Bokeh Visualization Library*. URL: https://bokeh.org/.

[6] Gilbert Tanner. *Introduction to Data Visualization in Python*. URL: https://towardsdatascience.com/introduction-to-data-visualization-in-python-89a54c97fbed.

[7] plotly. *Introducting Dash*. URL: https://medium.com/plotly/introducing-dash-5ecf7191b503.

[8] Plotly Dash. *Performance*. URL: https://dash.plotly.com/performance.

[9] Flask Caching. *Flask-caching*. URL: https://flask-caching.readthedocs.io/en/latest/.

[10] Chris Albon. *Python Machine Learning Cookbook*. 2018.

[11] Bolaji Ayodeji. *How to Write Good Commit Messages: A Practical Git Guide*. URL: https://www.freecodecamp.org/news/writing-good-commit-messages-a-practical-guide/.

[12] Steve Cornett. *Minimum Acceptable Code Coverage*. URL: https://www.bullseye.com/minimum.html#:~:text=Code%5C%20coverage%20of%2070%2D80,higher%20than%20for%20system%20testing..