# Final Report

## Interactive Data Visualization

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

TU Delft

# Contents

# 1 Problem Analysis

As data sets grow larger and larger, deriving valuable information from them becomes harder and harder. Not only because computations on the data become slower, but also because it becomes tough to know which data points expose a correlation or a causal relation with so many data points. In this chapter, we will analyze this problem and the essential correlated subproblems. Our goal is to create an efficient and extensible application that visualizes data into useful graphs. This visualizer should be interactive. When using it, the user should 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. The type of data the user can visualize and how they can interact with it is described in section 1.1. In section 1.2, we delve into the complications that such big data sets pose to the visualization process. And finally, the client has emphasized that building such an interactive visualizer should not be done from scratch unless doing so would otherwise be infeasible. In section 1.3, we further elaborate on this and on how the application should be extensible.

## 1.1. Data visualization

### Type of data

The application will be built to be able to handle time-series data of heart rhythm disturbance patients that are in a particular format. The client indicated they would 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.

### Nested filtering

To be able to graph, for instance, the data of 'female patients after 2017', one must be able to provide certain filters. And because one 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 it is essential that a convenient way to store intermediate results is found while memory use is minimized. A potential solution to this could be to cache the intermediate results.

### Thumbnails

Although graphs of many patients combined are very interesting, one might also want to inspect individual patient data. Therefore our application should show a thumbnail of a graph of individual patient data whenever one hovers over a data point of the corresponding patient in the original graph.

## 1.2. Computation intensity

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. Although important anyway, extra attention should be paid to this when expanding the project to a web application, since someone's browser may be very limited with regards to memory which might result in the application performing so bad that it becomes nearly unusable. Simply a tick of a filter box should not result in the whole data set being loaded in again.

## 1.3. Framework

### Libraries

Picking a library that provides the tools that help us to satisfy the clients' needs in the best way is a very important task. After discussing with our client we have concluded that we need either 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 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, they do prevent users from having to write a lot of functions and algorithms from scratch. This option is also a lot more error-prone. The other option, to use low-level libraries and build 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

The client has indicated that they should be able to extend the visualizer with additional charts, graphs, filters and/or similar things. Therefore, we have to create an object-oriented framework that makes it straightforward for our client to extend the visualizer with additional features in the future.

After the data set has been plotted, the user will need to be able to select a set of data points. The user can 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.

# 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. In section 2.1, we will elaborate on the process that was used to obtain the requirements. The resulting findings and general goal to aim for will be explained in section 2.2. Finally, in section 2.3, we will explain the MoSCoW method, which we used to list the functional and non-functional requirements followed by an actual list of them.

## 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 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 they can find the needs of our client effectively.

During the meeting with our client, we have noted the answers to 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 often quite useless. Therefore, our client would like an application that visualizes the patient data in an interactive way in which he can retrieve valuable information.

The data set size of patients which needs to be handled is 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, only the data of one particular patient needs to be plotted, and 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 data set 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 use 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 now follows.

### Non-functional

- The system will be able to select, filter and visualize data within 2 minutes.
- The system should be user-friendly to the extent 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 data set sizes of up to at least 5 gigabytes.

### 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 a 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 that 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.

### 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 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.
- The user should be able to give an order of loading when a data set is loaded.
- The user should be able to export and download figures made by the system.
- The system should avoid parsing the whole data set 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

### 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 colours for specific data sets.

### 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.

# 3

# Design and Values

This chapter presents the design for value approach used in the project. Great responsibility is required when handling the medical data of patients. In section 3.1, it will become clear why the data our application will deal with is very sensitive and what the consequences for an individual could be when the medical data would be leaked. Section 3.2 explains how leaks are prevented in our project.

## 3.1. Sensitive data

### 3.1.1. Privacy issues

Heart rhythm data of patients is very sensitive because the informational privacy of a patient will be compromised when the information is exposed. As stated in chapter 2, we are dealing with massive data sets on heart rhythm disturbance patients. The data contains aggregated data points of a lot of 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 someones medical record is 50 times more valuable than their credit card number. ”You can build entire human persona around a health record, seek or create medical treatment, abuse drugs or get prescriptions.” [2]

### 3.1.2. Exposing individuals

When heart rhythm (disturbance) data is 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 for instance their high heart rate or their heart rhythm disturbance condition. One could argue that since everyone knows smoking is bad for your health and they get 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.

### 3.1.3. Ethical questions

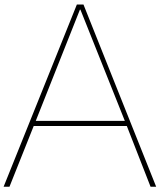
Some questions now arise: *is it fair to make people with heart rhythm pay more for health insurance? And what about people with heart diseases that could not have been prevented but are just ’bad luck’?*

These are questions that are very hard to answer, and there probably exists no perfect answer to them. It would be fairer 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, because deciding this is not up to us at all, we should do everything in our power to prevent our application from forcibly making such a decision. For instance by leaking all the data to third parties. Therefore we must treat the data very carefully.

## 3.2. Prevention of leaks

To prevent leakage of patient data, the application will run locally and on local data. 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, the application might be turned into a web application. A web application is a 'could-have requirement' from the client. 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 delivered 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.



# 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"*

# Reference List


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

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