Project Plan

Interactive Data Visualization

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

TU Delft

Contents

1 Problem Analysis **1**
- 1.1 Main goal . . . 1
- 1.2 Libraries . . . 1
- 1.3 Type of data . . . 1
- 1.4 Nested Filtering . . . 1
- 1.5 Functions and Algorithms . . . 1
- 1.6 Computation intensity and Thumbnails . . . 2
- 1.7 Extensibility . . . 2

2 Requirements Elicitation **3**
- 2.1 Process . . . 3
- 2.2 Findings . . . 3
- 2.3 MoSCoW Method . . . 3

3 Feasibility Study **4**
- 3.1 Technical Feasibility . . . 4
- 3.2 Operational Feasibility . . . 4
- 3.3 Schedule Feasibility . . . 4
- 3.4 Summary Feasibility . . . 4

4 Risk Analysis **5**
- 4.1 Sensitive data . . . 5
  - 4.1.1 Risk . . . 5
  - 4.1.2 Solution . . . 5
- 4.2 Lack of knowledge of libraries . . . 5
  - 4.2.1 Risk . . . 5
  - 4.2.2 Solution . . . 5

5 Project Approach **6**
- 5.1 Use of libraries and frameworks . . . 6
- 5.2 Efficiency . . . 7
- 5.3 Quality Assurance . . . 7
  - 5.3.1 Scrum . . . 7
  - 5.3.2 Documentation . . . 7
  - 5.3.3 Software Testing . . . 8
  - 5.3.4 Version Control . . . 8
  - 5.3.5 Static Analysis Tools . . . 8
  - 5.3.6 Definition of Done . . . 8

6 Solution Proposal **9**
- 6.1 Base solution . . . 9
  - 6.1.1 General . . . 9
  - 6.1.2 Libraries . . . 9
  - 6.1.3 Data manipulation . . . 9
  - 6.1.4 Data format . . . 9
- 6.2 Extensions . . . 9
  - 6.2.1 Web App . . . 9
  - 6.2.2 Alternative visualization . . . 10

7 General Planning **11**
7.1 Processes . . . 11
7.2 Tools . . . 11
7.3 Schedule . . . 11

A Requirements Questions **12**
A.1 General . . . 12
A.2 GUI . . . 12
A.3 Typer data . . . 13
A.4 Client/server division . . . 13
A.5 Non-functional . . . 14
A.6 Webapp (extra) . . . 14
A.7 Machine learning . . . 14

B Requirements **15**
B.1 Non-functional . . . 15
B.2 Must Have . . . 15
B.3 Should Have . . . 15
B.4 Could Have . . . 16
B.5 Won't Have . . . 16

C Schedule **17**

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

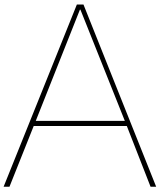
GitLab: For the project, we will collaborate by using GitLab. GitLab is also useful for keeping the issue board up to date. We will go over the issue board during every weekly meeting and assign the issues as a task to work on for the upcoming sprint. The issues may consist of bugs, new features, and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