Charles University

Faculty of Science
Bioinformatics

MASTER'S THESIS

Interactive clustering approaches in single-cell cytometry

Author: **Bc. et Bc. Nicole Aemilia Urban**

Supervisor: **Mgr. Adam Šmelko**

Academic Year: **2022/2023**

Declaration of Authorship

The author hereby declares that they compiled this thesis independently, using only the listed resources and literature, and the thesis has not been used to obtain a different or the same degree.

The author grants to Charles University permission to reproduce and to distribute copies of this thesis document in whole or in part.

Prague, January 27, 2023

Signature

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Mgr. Adam Šmelko, and RNDr. Miroslav Kratochvíl, Ph.D., for allowing me to work under their supervision on the the topic I am deeply interested in. I am also grateful for advice and encouragement I have received from them while working on the thesis.

Abstract

Flow cytometry allows inexpensive monitoring of large and diverse cell populations using fluorescent markers, providing immense applications in studying biological properties of blood and tissues as well as diagnostics in the clinical setting. Recent methodological advances highlight automatic clustering as a tool of choice for data analysis, and many clustering algorithms were developed for various use cases. However, the applicability of such algorithms in biology and medicine remains challenging unless the tools expose user-friendly, interactive interfaces that are accessible to domain experts. The goal of the thesis is to review the available methods that allow such interaction and supervision of the clustering process by the user, specifically focusing on interfaces desirable in clinical settings that do not require the user to interact with scripting or programming environments. As the main practical result, the thesis should design a new tool that builds upon previously developed methodology (iDendro, gMHCA), allowing the application of the researched methodology on realistic datasets. By using proper data visualization techniques, the end user should be able to interact with the dataset in a way that is both intuitive and useful for producing biologically relevant results. The thesis should also review data exchange formats that would be suitable for working with various other kinds of clustering algorithms.

Keywords interactive data analysis, visualization, cluster analysis, exploratory analysis, high-dimensional data

Author's e-mail urban.nicole.d@gmail.com
Supervisor's e-mail smelko@d3s.mff.cuni.cz

Abstrakt

Flow cytometrie umožňuje levné monitorování velkých a různorodých buněčných populací za použití fluorescentních markerů, díky čemuž je využívána jak ve výzkumu biologických vlastností krve a tkání, tak i jako diagnostický názor v klinickém prostředí. Nedávné posuny v metodologii zvýrazňují automatické clusterování jako nejvhodnější nástroj pro analýzu dat, a mnohé clusterovací algoritmy byly vyvinuty pro různé případy užití. Pokud však dostupné nástroje

nebudou interaktivní a snadno přístupné uživateli, praktické využití v medicíně a biologii nebude dostávat svého potenciálu. Cílem práce je zhodnotit dostupné metody, které umožňují interakci a supervizi clusterovacího procesu uživatelem se zaměřením na žádoucí rozhraní v klinickém prostředí, které nevyžaduje, aby uživatel interagoval se skriptujícími či programovacími rozhraními. Práce si klade za hlavní pracovní cíl nadesignování nového nástroje, který nastavuje na dříve vyvinutou metodologii (iDendro, gMHCA) a umožňuje aplikaci zkoumaných nástrojů na realistickém datasetu. Za použití technik z oblasti vizualizace dat by měl být uživatel schopný interagovat s datasetem který je zároveň intuitivní a užitečný pro produkování biologicky relevantních výsledků. Práce by také měla shrnout formáty výměň dat, které by byly vhodné pro práci s jinými typy clusterovacích algoritmů.

Klíčová slova interaktivní analýza dat, vizualizace, shluková analýza, explorační analýza, vysoko-dimenzionální data

E-mail autora urban.nicole.d@gmail.com
E-mail vedoucího práce smelko@d3s.mff.cuni.cz

Contents

List of Tables

List of Figures

Acronyms

xxx xxx

Master's Thesis Proposal

Author	Bc. et Bc. Nicole Aemilia Urban
Supervisor	Mgr. Adam Šmelko
Proposed topic	Interactive clustering approaches in single-cell cytometry

Motivation XXX

Hypotheses

XXX xxx

Methodology

Expected Contribution

Outline

1. Introduction: Explains thesis reasoning
2. Analysis of Flow Cytometry Data: Summary of relevant knowledge regarding flow cytometry and data analysis
3. Automatic clustering:
4. DESCRIPTION OF MY WORK: find a different name for this section
5. Discussion and limitations: Thesis limitations are addressed
6. Concluding remarks: Findings and implications summary

Core bibliography

S. Das, B. Saket, B. C. Kwon, & A. Endert, "Geono-Cluster: Interactive Visual Cluster Analysis for Biologists," in IEEE Transactions on Visualization and Computer Graphics, vol. 27, no. 12, pp. 4401-4412, 1 Dec. 2021, doi: 10.1109/TVCG.2020.3002166.

Sieger, T., Hurley, C. B., Fišer, K., & Beleites, C. (2017). Interactive Dendrograms: The R Packages idendro and idendr0. Journal of Statistical Software, 76(10), 1–22.

Šmelko, A., Kratochvíl, M., Kruliš, M., & Sieger, T. (2021, September). GPU-Accelerated Mahalanobis-Average Hierarchical Clustering Analysis. In European Conference on Parallel Processing (pp. 580-595). Springer, Cham.

Fišer, K., Sieger, T., Schumich, A., Wood, B., Irving, J., Mejstříková, E., & Dworzak, M. N. (2012). Detection and monitoring of normal and leukemic cell populations with hierarchical clustering of flow cytometry data. Cytometry Part A, 81(1), 25-34.

Author

Supervisor

Chapter 1

Introduction

XXX

Chapter 2

Analysis of Flow Cytometry Data

The following chapter introduces the reader to basic information about flow cytometry, its origin, underlying mechanisms, and its use in academic and medical settings. The reader is acquainted with flow cytometry data analysis techniques.

2.1 Introduction to Flow Cytometry

Flow cytometry is a high-throughput laboratory technique that allows for studying cellular populations, and is used for hypothesis testing as well as in the medical areas in both biomedical research and diagnostics, most importantly in clinical immunology. As described in Black *et al.* (2011), cell-based screening also facilitates the development of safer and more effective drugs.

Flow cytometry is non-destructive.

Goals Flow cytometry aims to analyze the physical and chemical features of small particles. In a biological context, the particles are usually various cells, although they can also be bacteria (Steen, 2000; Nebe-von Caron *et al.*, 2000). By using the flow cytometry technique, one can expect to see the division of input cell population according to various parameters, such as extracellular vesicles (Nolan, 2015), membrane proteins (Schmitz *et al.*, 2021), antibodies (Kalina *et al.*, 2020; Hall & Rosse, 1996), and intracellular particles (Pirone *et al.*, 2021; Wrońska *et al.*, 2022). Using flow cytometry for measurements of molecular interactions such as ligand binding (Nolan & Sklar, 1998) or protein phosphorylation state (Perez & Nolan, 2002) is possible.

Flow cytometry has facilitated access to new intracellular pathways, which were not revealed by other biochemical approaches (Sachs *et al.*, 2005).

History Flow cytometry was developed to suit the need to analyze cells and other particles. The prototypal development started in the 1940s with the invention of a device known as Coulter counter. The Coulter counter was a small device that relied on particles flowing in a conducting fluid. The movement was directed by an electrical field, which produced a measurable electrical impedance, through which it was possible to measure the size and number of cells Graham, 2013. The idea has been improved ever since then, focusing mainly on the three underlying physical systems: fluidics, optics, and electronics. Each of the improvements led to more precise measurements (Picot *et al.*, 2012). There were many growth opportunities for the Coulter counter, such as the development of a multichannel solution or the replacement of the analog electrical circuit with a digital one. Flow cytometry was built upon the foundation of the Coulter counter.

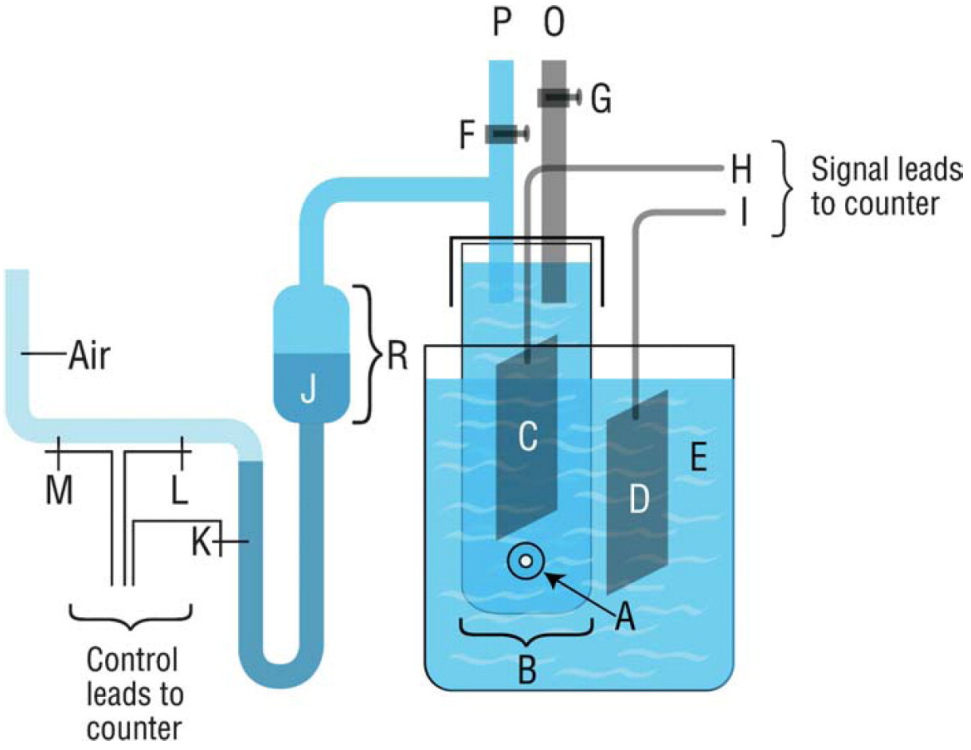

Figure 2.1: Schematic of the Coulter sample stand taken from Don (2003)

Principle Three main physical systems come into play: fluidics, optics, and electronics.

Fluidics is responsible for directing the suspended particles through a flow cell, in which the optical part of the cytometer operates. The flow of the particles needs to be controlled, steady and predictable. The cells need to be sorted and directed. A well-designed system can leverage fluidics to prevent clogging and blockages. Today, one can talk about microfluidics rather than fluidics. Microfluidics operates at much smaller scales, bringing the necessary dimensions down to micrometers, promising a new, more portable and precise generation of devices (Gong *et al.*, 2019).

Optics is responsible for creating scattered light or emitted fluorescence, which carry information about the cells count, granularity, or chemical composition. Parts of the optical system are lenses that shape and focus lasers beams and lasers to provide a source of light (Adan *et al.*, 2017).

In the past, the lasers were noble gas lasers, and that meant mostly argon-ion lasers (Kamentsky & Kamentsky, 1991). Later, cytometers were updated so they can take advantage of solid-state lasers. The advantage of lasers over different light sources is that the illumination point can easily be focused. The light scatters of individual cells flowing in a stream, targeting small sample volumes at a time, leading into sorting cells by size and complexity, cell cycle, or cell viability. Cells can be tagged with dyes or antibodies (Wilkerson, 2012). The usage of multiple lasers is possible (Bigos *et al.*, 1999; De Biasi *et al.*, 2016; Ashcroft & Lopez, 2000). A comprehensive overview of lasers in flow cytometry can be found in Shapiro & Telford (2018).

Electronics is responsible for detecting signals from flow cytometry. A useful tool is called discriminator, which is a circuit that can be aimed at specific voltages thus preventing circuit overload. Amplifiers are used to produce signals, translating the input into information. The electronic systems can take advantage of more complicated parts as explained in Robinson (2004).

Tools A device used for the flow cytometry technique is called a flow cytometer. Flow cytometers can go through thousands of samples a day.

Commercially sold flow cytometers can be sorted into two categories: analysers or sorters. As the name suggests, sorters both collect data and sort cells by their properties. Flow cytometers can also be adjusted to specific use cases, such as the Ploidy Analyser manufactured by Partec, which is mainly targeted at analyzing plant material as plants are regularly polyploid - meaning they

have at least three complete sets of chromosomes (Zhang *et al.*, 2003; Jacob & Pierret, 1998; Geng *et al.*, 2011).

However, improvements such as the development of new fluorescent dyes led to a massive increase in dataset size. Having more colors at one's disposal allows for tracking more parameters at a time, creating a subtechnique of flow cytometry known as polychromatic flow cytometry. ???? intracellular parameters, producing rich information about individual cells (Wood, 2006), that is impossible to process in traditional manual ways such as sequential manual gating, which is also observer-dependent and requires high specialization, often impacting results. It is also incredibly time-consuming. The lack of a fully functional automated tool hinders the full potential of flow cytometry.

To deal with the high-dimensional data and the issue of evergrowing time and space requirements, computer-driven analytical techniques have been introduced, especially but not only from the area of hierarchical clustering as described in section 3.1.

Methods that are reliant on prior knowledge of expected cell populations (clusters) (Lo *et al.*, 2008; Rogers *et al.*, 2008; Wilkins *et al.*, 2001; Zeng *et al.*, 2007) to deal with automatization and move the thing to the less observer-dependent manner are often.

2.2 Analysis

Several methods of flow cytometry dataset analysis will be described in this section. The analysis of data obtained from a flow cytometer can be done manually with all of its possible downfalls, or with the help of either commercially or freely available software.

As with other types of workflows, quality assessment and control is essential. Through quality assessment, one can make sure that the differences among samples are biological, not technical. One of the approaches which can solve the issue is the development of graphical tools which can reveal non-biological differences among samples (Le Meur *et al.*, 2007). Other methods suggested calibrating computation for each of the fluorescent parameters (Gratama *et al.*, 1998).

Gating Gating is a method that creates subsets of particles based on the flow cytometer output. Gating works by using defined regions called "gates". Gates can be of various dimensions and shapes.

Sequential gating is a subtechnique of gating and is useful for the identification of specific populations. Sequential gating differs from regular gating in terms of the number of used gates and the order in which the gates were used. In sequential gating, each gate refines the population selection, starting with removing background noise. Gates can be added or tweaked in order to include or exclude data based on various parameters such as size or fluorescence intensity. Alas, it is limited in its visualization capabilities, as it can only take one as far as two parameters simultaneously. It is also prone to observer-dependent errors (Herzenberg *et al.*, 2006).

Due to the algorithmic nature of gating, it's a process that can be automated. Automation has a couple of advantages including human error prevention. However, automated gating difficulty increases with nonconvex cell populations, meaning populations that are neither concave nor convex but rather they curve up and down, or other multidimensional shapes, and can struggle with elliptical shapes that are often produced from flow cytometry Finak *et al.* (2009).

Probability Binning Probability binning can help identify differences that are not visible through sequential manual gating by distributing the data into bins of the same size and comparing the count of events between the experimental sample and the control sample. While historical methods that include Overton subtraction (Overton, 1988) or Komogorovs-Smirnoff statistic (Young, 1977) were useful, they suffered from enormous counts of bins. Probability binning works by minimizing the maximum expected variance by creating bins of various sizes. The smaller the bins, the more events, however, at the end, each bin no matter its size houses the same amount of events. It then compares the distributions and does not require setting the expected number of cell populations from the beginning. One of its main advantages is that its computation time does not increase with scaling to more parameters. With quality assessment, it can escape one of its major downfalls, which is the sensitivity to variation in experimental conditions.

Frequency difference gating (also known as frequency subtraction gating) takes advantage of probability binning and aims to find and identify rare populations. An example application of frequency difference gating can be found in Roederer & Hardy (2001).

Cluster Analysis Due to the nature of this thesis, cluster analysis is described in depth in 3.1. One does not have to work with high-dimensional data when they can get rid of the dimensions via principal component analysis. Principal component analysis is an unsupervised method of dimension-reduction, which works by creating a new dataset in which the new variables - principal components - are linear combinations of the original values. This allows for detecting patterns (Räuber *et al.*, 2021).

Both principal component analysis and cluster analysis also have the advantage of being able to take fluorescent values individually and therefore account for individual variation.

Chapter 3

Clustering

Clustering is a method of finding structural patterns in given data and sorting the data points into groups or subsets. In general, data points in one cluster are more similar to each other than to data points in other clusters. Clustering can be done in either supervised, semisupervised, or unsupervised fashion.

Cluster analysis is a collection of various algorithms and methods that can be used in said grouping process, with each being appropriate for different jobs. The basis of each model is different as well, using various measures such as distance (Murtagh & Contreras, 2017; Hsu *et al.*, 2007). The basic overview can be split into connectivity models (Fischer *et al.*, 2003), centroid models (Morissette & Chartier, 2013; Sun *et al.*, 2014), distribution models (Jiménez *et al.*, 2019; Bocchieri & Mak, 2001), density models (Campello *et al.*, 2020), and other. This thesis focuses mainly on hierarchical clustering, which falls under connectivity models and uses distance as its metric of choice.

3.1 Hierarchical clustering

Hierarchical clustering is one of the possible approaches to classification problems. It is a tool that gives an idea of the data structure, and allows for association between subclusters while also keeping a level of distinction. Hierarchical clustering can be done in two ways: bottom up, also known as agglomerative, in which the displayed distance is of two clusters, and top down, also known as divisive, in which the distance displayed is between individual observations. Only the agglomerative approach will be discussed in this thesis.

In the agglomerative approach, two closest elements are conjoined into a cluster, which is then joined into a larger cluster with the closest element. In

this iterative way, a singular final cluster is made and a hierarchical overview of the data is formed. In this way, an agglomerative hierarchical algorithm displays the similarity between clusters and only check distances between data points.

""" In hierarchical clustering, the merging points of clusters are called "nodes" or "linkage points". Each merging point is represented by a node in the dendrogram, which is the graphical representation of the hierarchical clustering. A dendrogram is a tree-like diagram that shows the hierarchical structure of the clusters at different levels of granularity. The leaves of the dendrogram represent individual data points and the branches represent clusters of data points, which are formed by merging smaller clusters. The linkage point show the distance between the two clusters that are being merged, and the height of the linkage point in the dendrogram represents the distance or similarity between the merged clusters."""

3.2 Distance

Distance can never be a negative number, and scales from 0 for objects that are not different at all, to larger values for more different objects. The default distance metric is usually Euclidean, although in flow cytometry context, the usage of Mahalanobis distance instead could be a better idea Fišer *et al.* (2012).

Euclidean distance is a distance metric used between two points in spaces of higher dimension than 2. It can be found by using the Pythagorean theorem and solving for hypotenuse.

$$x^2 + y^2 = z^2$$

ADD d(p, q) = root((p1-q1)$\hat{2}$ + (pn-qn)$\hat{2}$))

It can also be found using polar coordinates:

ADD d(p,q) = root(r$\hat{2}$ + s$\hat{2}$ - 2rs cos(theta-phi))

Mahalanobis distance is useful for keeping both orientation and axes ratios while serving as an expansion factor for the best fitted ellipsoid, allowing for more efficient processing of elongated clusters Zamir *et al.* (2005). It is a distance metric distances used for distance between a point P and distribution D.

It follows this algorithm:

1. find centroid/center of mass

2. Estimate standard deviation of distances between points and center of mass

3. If distance(test point, center point) ¡ 1 stdev -¿ point belongs to the set with a high pobability

4. Plug the above, which can also be written as

$$(test_point - sample_mean)/standard_deviation$$

Chapter 4

Data Visualization

Data visualization is crucial for human friendly work and allows for simpler interpretation of the results, various tools and approaches have been developed. Visualization is one of the core parts of data science. It allows not only for the data scientist to understand the data and the oucome of their work better, but also to share it with the shareholders or the public in a clear way.

The cluster structure can be graphically represented in a dendrogram, which by its nature consists of n - 1 pairs of branches over n observations, each branch representing one split. The distance between sub-clusters is given by the height of the branches.

However, it is not enough to have the dendrogram generated. The user might also want to be able to interact with the results, zooming in and out of parts, choosing various colors for different clusters, or selecting only parts of the final dendrogram. Tools for dendrogram visualization and inspection have been made as described in Sieger *et al.* (2017).

Dendrograms are often paired with heat maps, another data visualization tool in which values are represented in a matrix of values, that are later translated into colors. Color can also be assigned to each cell population, distinguishing them by their markers Ellyard *et al.* (2019). Advanced heatmap tools have also been implemented in R (Zhao *et al.*, 2014). Patterns of large datasets can be displayed via heatmaps without the need to analyse the data first (Key, 2012).

Chapter 5

Overview of available tools

Many solutions have been made to ease the exploration, analysis, and visualization of data. The solutions can be either paid (Tableau, FIND EXAMPLES) or available as freeware (Google Charts). However, these solutions often are not suitable for bioinformatical purposes.

There are many options for programming languages that can be used for data visualization (R, Scala, Matlab, Python, Java, C#, and others). Python and R are especially popular in bioinformatics and biology (Giorgi *et al.*, 2022; Gentleman, 2008; Bassi, 2016). There are many advantages to using Python. The user base is rather large - its userbase was the 5th largest on StackOverflow (Srinath, 2017). Its learning curve is flatter compared to other languages. Python also has the ability to follow multiple programming paradigms (object-oriented, imperative, functional, and for more experienced users even procedural) (Srinath, 2017; Dyer & Chauhan, 2022). Python is dynamically typed, making prototyping easier at the cost of performance (Tratt, 2009). There is also an abundance of free-to-use libraries.

As stated in **??**, the ability to visualize the input and output of one's work is crucial to communicate patterns and information across in a clear and intuitive way. Choosing the right tool for the job is crucial.

In this thesis, several mostly Python frameworks for building both front-end apps and graphing solutions are tested and compared. Keep in mind that the user should not look for one "best" solution and rather analyze their needs and expectations and choose a tool accordingly.

5.1 R Shiny (ShinyDendro)

R Shiny is the only non-Pythonic open-source package that is discussed in the thesis. It is used for building web applications aimed at data scientists, as the tool is used for data exploration, analysis, and visualization. The finished application can be run either locally or deployed to a server, making it accessible to users all over the world. No knowledge of web development stack - HTML, CSS, JavaScript - is necessary. The R side of things is responsible for the data processing and analysis, while the rest of the stack handles user interface and interactions. A variety of premade tools is available, such as buttons, sliders, or menus. Premade layouts are also available, speeding up the process to a well-rounded looking application.

In 2022, PyShiny was announced, bringing Shiny to the Python ecosystem.

5.2 Streamlit

Streamlit is an open-source solution designed to make the creation of web apps as accessible to Python users as possible. The framework was created with machine learning and data science in mind.

Streamlit does not require any knowledge of other languages that are usually used for web apps (Javascript, CSS, HTML, and others), and one of the huge advantages is that it removes the necessity of data transmission between Python and a non-Pythonic front-end. However, knowing HTML and JavaScript allows the user to develop their own Streamlit Components. The goal of Streamlit is to bridge the distance between Python and React, one of available JavaScript libraries, which is often used for creating interactive frontend interfaces, effectively wrapping React components. It is supported by Windows, Mac, and Linux operating systems, and by December 2022, it has over 21K stars on GitHub. It has demo apps published on GitHub to get one started. As per usual with Pythonic packages, installation is easy via a pip command, and prototyping can start fast. It brings its own syntax, but the learning curve is not too steep.

Streamlit has been built to seamlessly integrate with matplotlib, plotly, seaborn, or even altair. It's also compatible with OpenCV, Vega-Lite, LaTeX, and many more. Streamlit is not a tool useful to anyone who would like to use Python Notebooks.

Streamlit can help the user create impressive dashboards that help to com-

municate data story across, allowing for exploration, analysis, and reporting. The fact that Streamlit is compatible with multiple graphing solutions means they can be joined on one dashboard, taking advantage of each of their individual strengths and weaknesses.

Another major advantage of Streamlit is how it can be used on an already existing code base.

However, as with any solution, Streamlit comes with some disadvantages. For example, Streamlit does not allow for much customization, and changes that might be desirable and seemingly easy, such as having pop-ups in one's app or changing the visual of one's buttons, are simply not possible in native Streamlit environment. Streamlit also comes with a limited data upload which caps at 50Mb. Other methods such as getting url parameters require hacks that are not scalable for large operations (github issue 798).

The support for animation and video is also fairly limited, as is the interactivity of graphs.

Any time a change is made, the whole program reruns, and everything is recalculated, bringing potential performance issues. It does provide a caching solution, which is unfortunately not bullet-proof, and the user might find long loading times off-putting in case of the page loading over several minutes. The user is also not notified visually when the app is recalculating, which once again might be difficult for apps with long loading times.

Natively, Streamlit requires the user to run it with an external shell command. The user has to be sure that all of the dependencies are installed, and some technical knowledge is required to start the local server.

5.3 Dash

Dash is another open-source Python package designed for building web applications. As the name suggests, Dash's main focus is on creating dashboards. Just as Streamlit, it can be installed via a simple pip command, and supports other languages used in bioinformatics such as R or Julia. The provided API is enough to work with for a Python user with no web development knowledge (HTML, CSS), once again though, knowledge of the web development languages can hugely benefit the user. Flask is used for Dash backend, and provides a Python wrapper for front-end part of the application, and Dash is said to be production ready. The high level API can create charts similar to what D3 is capable of. React.js is used for component rendering.

With just under 18K starts on GitHub in December 2022, it's just a tad less popular solution on the platform than Streamlit. However, Dash is a more mature product, which has been around for longer. More questions have been asked and answered on the internet.

Dash outperforms Streamlit in both customizability and performance. It is less opinionated design wise. The documentation is well done, and kept up to date. The callback debugger is useful. HTTP Basic Authentication is provided.

With the release of JupyterDash, integration with Jupyter is supported, even though it can be a bit unstable at times.

Dash also comes with a wide variety of example apps, which are available in the so called Dash App Gallery.

Since Dash comes from Plotly, it is mainly designed to work with Plotly graphing solution. Using other graph libraries is possible but it can be a bit of a hussle.

One of the core features is the reactive Dash callback decorator. Plenty UI elements can be changed via the callbacks (dropdown menus, sliders, graphs), and the whole application has to be set accordingly to work with the linked input. Dash requires a bit more biolerplate code than R Shiny, however, it is easier to integrate one's CSS with Dash than R Shiny.

Dash is not suitable for building large scale web applications, and web applications with a lot of functionality other than analysis should not be expected from it. Proprietary components have to be written in React.js.

PROBABLY ADD SECTIONS DIVIDING GRAPHING SOLUTIONS AND FRONTEND

5.4 PyScript

PyScript is a framework that promises connecting Python code to html, letting applications run in the browser without any JavaScript knowledge. JavaScript is not a programming language of choice for many scientists [SOURCE]. It requires little to no initial setup. The creation of PyScript was enabled by Pyodide, which is a CPython way to WebAssembly.

PyScript is added to one's application by adding a link to it's source at the head of the HTML file. This solution requires a working internet connection, however, the files can be downloaded and linked locally, making it possible to run offline. PyScript comes with a .js file and a .css file.

It is preferred using the Python code as a source, rather than inserting the code directly into the HTML PyScript tag to prevent potential formatting issues.

Some Python libraries are available and need to be specified under a py-env tag or install via micropip.

It is possible to interact with several HTML elements such as document, window, or console. Alternatively, it is possible to leverage Pyodide to create proxies between Python and JavaScript callbacks, or use prepared UI components such as buttons and text boxes.

One main difference between PyScript and other Python solutions such as Dash 5.3 or Streamlit 5.2 is that while the latter solutions require a server to run, PyScript operates on the client side. Unlike Dash and Streamlit, basic HTML knowledge is necessary.

It was announced in 2022, and as of 2023, it is in alpha stage of development. With that said, core functionality is still under development and may change in the nearby future. Additionaly, it lacks a lot of functionality one would need from such a solution, such as Plotly 5.9 not being supported. However, alternatives such as Bokeh are available. Even so, the graphs are not interactive.

Pyscript makes it more difficult to debug one's application.

PyScript was not chosen as a solution for this thesis as its loading times are incredibly slow [TIME IT, maybe Hello World?].

5.5 Altair

Altair is relatively constrained as a graphing solution. It was build on Vega-lite, which calls itself a visualization grammar, and is a declarative language for interactive visualizations. While Altair in its core uses JSON to define properties, the user of Altair Python library does not need to do such thing: Altair does the converting for the user based on the inputs. It uses Pandas, therefore data can be manipulated in a manner similar to Pandas. Apart from Pandas DataFrame, the user can choose to input data in other formats such as Data and related objects, json file, csv file, url pointing to either of previously mentioned files. Altair is also of declarative nature meaning that Altair chooses a lot of "how to do" for the user and the user can focus on the "what to do", with a high-level approach. As other Python libraries, it can be easily install via a pip command.

Altair wishes to keep the user focused on the data rather than formatting.

In general, it can create simple graphs in less code than let's say Matplotlib, and much of the visual side of plots are taken care of by the Altair package.

One of the biggest Altair advantages is that it comes with interactive plots out of the box, and the user can immediately take advantage of desirable functions such as zooming in and out, or highlighting parts of the plot. It's also extremely easy to connect multiple plots and have the selection apply to all of the graphs.

Altair takes advantage of marks, which provide basic specification of visual representation of data, such as bar, image, circle, rectangle, and so on as provided by Chart.mark* methods. Once the data is input and the mark is chosen, the user can call the encoding. Encoding takes care of the placing of the chosen representations, setting the axes, choosing the colors to show individuality of the data points, the opacity and else. Encoding expects simple answers about the variable type. This setup promotes code reusability, in which the core code can be left the same while tweaking the plots.

By chaining more commands like interactive, tooltip or selection, the user can make Altair perform a variety of actions without setting up much of the code.

From graphical point of view, one can write functions in Altair that are re-usable, for example a function for adding text with specific parameters can be defined once and used multiple times. This ability is connected to using Altair objects as return types and such use is recommended.

One of the bigger disadvantages of Altair is the size constraint, and Altair documentation itself recommends to use only data with 5000 rows or less in order not to run into issues. In practice, this results into raising the MaxRowsError, not allowing the user to go over 5000 rows. This behavior can be disables, but as mentioned before, that is not recommended. Altair also does not support 3D visualizations, although a dimension can sometimes be added to a 2D graph with clever use of color. The support for statistical plots is limited, and there are better solutions to use when trying to plot e.g. linear regression. Compared to other plotting libraries, Altair is slow.

5.6 Networkx

Networkx was decided to be unfit for the task as while representing dendrograms in a form of a graph is possible, it is not the best solution. –maybe its not possible?

5.7 Matplotlib

Matplotlib is a famous Python package based on MATLAB. It offers plenty of functionality. As it's based on MATLAB, the interface will seem familiar to those skilled in MATLAB, and it is both object oriented and state-based, which can lead to confusion at times.

Matplotlib has been around for quite some time, and has amassed over 16.5K starts and over 650K users on GitHub as of December 2022. Having a huge user base is an advantage, as there are usually more resources available. The documentation provides plenty of real life examples, and many more can be found scattered across various user-made applications.

The low-level interface can be considered both an advantage and a disadvantage, for there is a learning curve. However, it also makes plotting even the most wild and complicated plots possible with a bit of work. Working with large and complex datasets can be challenging regardless of the chosen library.

Matplotlib integrates flawlessly with Numpy, sklearn, or pandas.

Creating 3D visualizations is possible in Matplotlib. Matplotlib comes with an animation module, bringing life to the user's graphs. The data representations can be either static, interactive, or dynamic. In general, there are better options for representing time series than Matplotlib.

5.8 Bokeh

Bokeh is also on a list of available open-source Python libraries that make data analysis easier. It cooperates with more web focused mediators such as Streamlit or PyScript. The API for creating plots and visualizations is designed to be simple and intuitive, and allows to be displayed in web browsers with use of JavaScript. The customizability is high, many details about the plots can be changed, including visual details such as color and line styles. The integration with Jupyter Notebook is supported, and Bokeh also gladly cooperates with Flask or Django.

Although the API is quite simple, the underlying JavaScript can be overpowering for developers and data scientists who lack experience with web development.

Apart from traditionally expected types of visualization such as line plots, scatter plots, or bar charts, it also comes with built-in tools that allow the user to interact with the plot. However, as Bokeh is a high-level tool, users might

struggle to find the amount of customization they desire. Bokeh also does not go well with 3D visualizations.

Unlike Altair, Bokeh's strenght comes from the ability to handle large datasets. It also allows the user to link multiple plots together and other desirable options expected to be necessary in a data exploration tool.

5.9 Plotly

Plotly is also an open-source Python package. Plotly is built on plotly.js, which is a Plotly JavaScript library, and also uses tools such as D3.js, HTML, and CSS. As such, it is appropriate to use with web in mind. Plotly also supports Jupyter Notebook integration. Plotly is also a high level declarative tool, meaning that it takes care of large portions of the visualization if the relationships between the data are specified clearly. While declarative tools save user time and code lines by making a lot of decisions for the user, it comes at an expense of less control over the plot and the details overall.

Plotly can be used for interactive graphing, it supports 3D charts, and many other expected types of plots (scatter, violin, box, and others). Plotly offers tools for interacting with the plots, such as the hover feature, which can be used to identify outliers. However, the sheer amount of tools Plotly comes with sharpened learning curve and decision paralysis.

Multipanel layouts are available, and while linking the panels together might not be as straightforward as in Altair, it is not overly difficult.

Chapter 6

DESCRIPTION OF MY WORK

The following chapter discusses the process of creating an interactive visualization tool, choosing the right frameworks, and using them to create new solutions to explore flow cytometry data. The importance of skills in various fields such as data science, statistics, design, and software development is highlighted. Architecture of final solution is described.

6.1 Requirements for developing a visual analysis tool

While each of the frameworks described in 5 has its own merit and its own pitfalls, the overall performance of a tool depends on user's requirements, expectations, and skills. Building data visualization tools is based on a variety of fields, such as data science, statistics, design, and software development.

Skills from the area of design are understanding the use of color, typography, and layout, which can all help communicate insights about the data clearly to the end user.

From software development side of things, one has to be able to use a language of choice to enable the implementation of concepts from the past paragraph, and also broad knowledge of available tools and libraries is welcome. The implementation does not have to be as straightforward as it would seem at first glance, so strong problem-solving skills are welcome, as is orientation to detail.

6.2 Selected Technologies and Architecture

Two frameworks described in 5, namely Streamlit 5.2 and Dash 5.3, were chosen as frameworks in which the visualization tool would be developed. The rich communities of both frameworks provide a deep pool of knowledge and promise support from the creators and future development. Both have proven to be tools worthy of the task of visualizing and exploring a flow cytometry dataset.

The tool consists of several layers as described below.

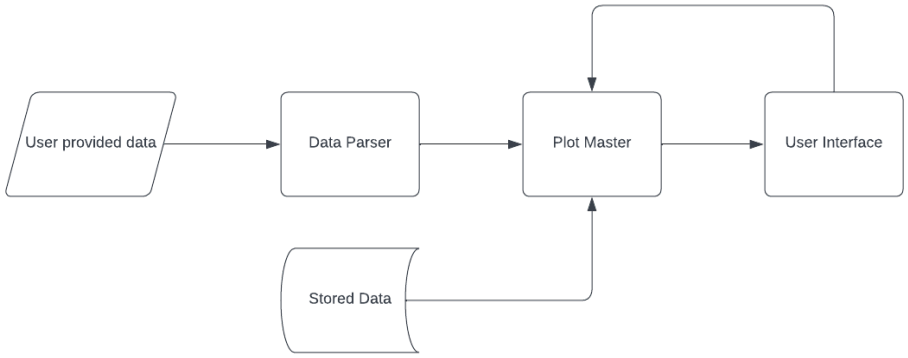

Figure 6.1: Flowchart

6.2.1 Data Parsing

A custom parsing solution was developed to read and translate input data necessary for dendrogram visualization. The input data is expected to be put in the "user_data" folder. Three files are expected:

- heights.csv

  Heights refer to the height attribute used in dendrograms. The height attribute represents the heights of individual nodes in the dendrogram in the form of a numeric vector. Each element of the vector is tied to a specific node. Each of the elements measures height from the beginning.

- merge.csv

  Merge is a 2D matrix that describes the hierarchical clustering solution. Each row is a description of two components used in a merger. Positive values indicate individual observations (leaf nodes) and negative values indicate clusters formed in previous mergers. More on the iterative process of hierarchical clustering can be read in 3.1.

- order.csv

  Order is a vector of positive integers that describes the order in which observations were merged together. The indexing of the data in order.csv is expected to start at 1, as DataParser subtracts 1 from each element to match Python's indexing, which starts at 0. Order is useful for data visualization as it helps the program navigate the easiest way to plot leaves without creating unnecessary cross-overs.

It is also possible to add a fourth file:

- labels.csv

  Labels is an optional file that should contain a 1D matrix of strings describing each of the datapoints. However, in case the user does not have labels available, mock labels will be provided assigning each element a value from 0 to len(data).

The file format was inspired by the output of R's hclust function.
One file that is not used in plotting the dendrogram is:

- data.csv

  The data.csv should contain a matrix in which columns represent features and rows represent each of the samples. It can either be raw or preprocessed data. This file is used to plot data represented in dendrogram with the aim to make data exploration more accessible.

6.2.2 Data Plotting

The main component responsible for plotting the data is PlotMaster. The PlotMaster expects data which has been preprocessed by the DataParser. This architecture was selected for merits of having each of the processes split, making it easy to build prototypes in multiple front-end solutions. A well ordered application reduced confusion should the user need to understand anything from the code.

The PlotMaster leverages Plotly (for deeper description of Plotly see 5.9) graphing library under the hood. It is designed to help the user explore the data in a visual way. The ability to pan, zoom in, zoom out, or save the plot as .png file is integrated into Plotly.

There are four types of plots offered by the PlotMaster:

- Dendrogram

  Dendrogram is a tree representation of hierarchical clustering as described in 3.1. The user can interact with the dendrogram by setting color threshold. Color threshold is an integer which is responsible for the height in which clusters are set to different colors, helping the user detect them visually.

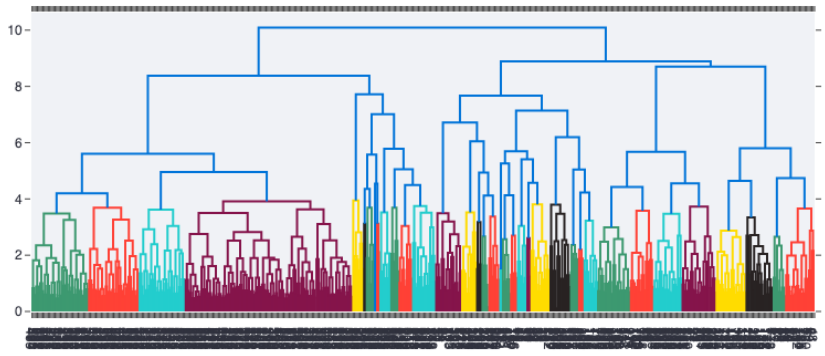

Figure 6.2: Dendrogram

- Heatmap

  Heatmap is a way to visualize data quantity in color dependent manner as described in 3.1. The user can specify which features they would like to plot in the available heatmap.

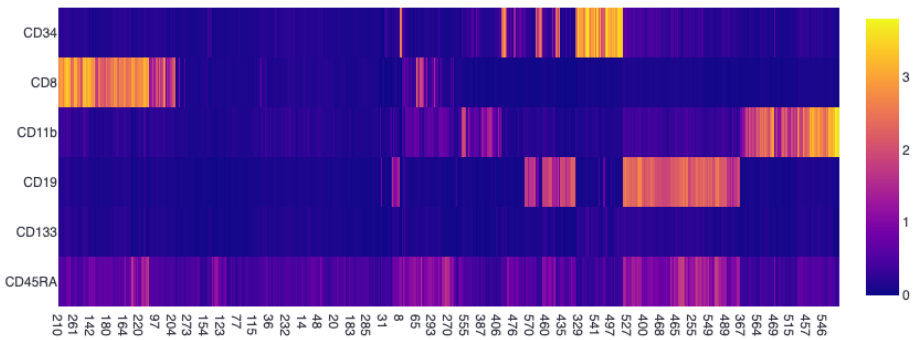

Figure 6.3: Heatmap

- 2D exploration of data

  The 2D exploration of data offers several options: the selection of two features, scatterplot matrix for all features, and various dimensionality reduction algorithms (PCA, t-SNE, UMAP). The color of each of the datapoints matches the color in the dendrogram. Scatterplot matrix for all features is not recommended for large datasets.

  The PlotMaster searches for data reduction data in the "user_data" folder. Should the data not be found it is created by leveraging the sklearn library, and saved in the "user_data" folder for later use.

Figure 6.4: Two Selected Features

- 3D exploration of data

  The 3D exploration of data focuses on three dimensionality reduction algorithms: PCA, t-SNE, UMAP. The color of each of the datapoints matches the color in the dendrogram. The user can rotate the plot at will.

  The PlotMaster searches for data reduction data in the "user_data" folder. Should the data not be found it is created by leveraging the sklearn library, and saved in the "user_data" folder for later use.

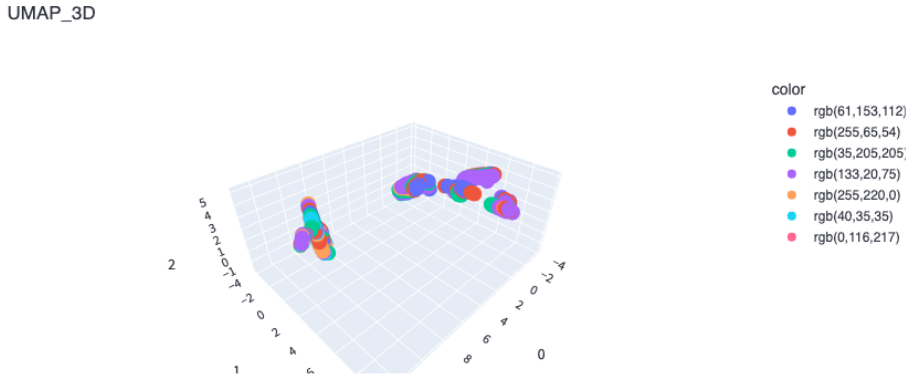

Figure 6.5: UMAP 3D

6.2.3 User Interface

Streamlit The interface in Streamlit was designed to be simple. Streamlit comes with some formatting out of the box, therefore it is easy to plug it in and have a decent looking application immediately. It also has naive support for Plotly, making it easy to connect the interface to the plotting part of the application.

For layout, the page configuration was set to wide to make most of users screen's real estate, with two columns. The layout scales with Brave and Chrome browser zoom tool.

On the left, there is a slider, a dropdown menu, and two graph: a dendrogram and a heatmap. The slider is connected to the dendrogram's color threshold, and the user can select any integer from 0 to the maximum value in heights.csv. The dropdown menu allows for selection of multiple values. The values are strings representing features in the dataset. Selected options are plotted in a heatmap, which can be found just under the dendrogram.

On the right, there is a dropdown menu which allows for selection of multiple options. The offered options are tied to 2D and 3D plots described in 6.2.2. Plots are displayed in selected order.

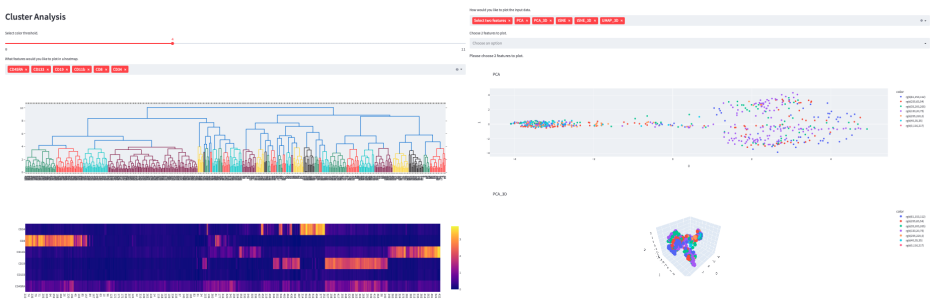

Figure 6.6: Streamlit Layout

6.3 Architecture Challenges

Plotly Even though Plotly presents itself as a plotting library, it takes care of more than just plotting data. For example, plotly offers a function called create_dendrogram, which offers nice tools for working with dendrograms, such as assigning colors, giving reports about each of the leaves, and more. However, the function expects input in the format of raw data, which is incompatible with the input as required from the user (described in 6.2.1).

To work around the issue, the function was monkey patched. Monkey patching allows to modify or extend behavior at runtime, eliminating the need to alter the original source code. Thanks to monkey patching the function and underlying Dendrogram class, it was possible to plot the dendrogram data that were obtained from a different algorithm than the one hiding under create_dendrogram.

Streamlit One of the main challenges when working with Streamlit (5.2) is the lack of intelligent interaction with the interactive components provided by the solution. Any time a change is made anywhere, the whole code is rerun, prolonging waiting times for the user. That is also true for components that were not affected by the change.

Streamlit offers a couple of solutions to the issue. It is recommended to precompute and everything possible, and load the precomputed files into Streamlit instead.

Another option is to leverage Streamlit's naive cache. The cash promises improved manipulation with large datasets and performing expensive operations. It can be implemented by using the @streamlit.cache decorator on expensive functions. When a function is called with @streamlit.cache decorator, Streamlit checks whether the input parameter has changed, whether any value of any

external variable has changed, what is inside the body of the function, and the body of any function called inside the decorated function. Everything is cached the first time the program is run, making the first run more expensive than subsequential runs.

Apart from cache, Streamlit also offers a tool named experimental cache primitives. Experimental cache primitives aims to ameliorate caching, as @streamlit.cache has been deemed "hard to use and not fast" (according to Streamlit's official documentation). As a fix, Streamlit followed development with the addition of two more decorators: @streamlit.experimental_memo and @streamlit.experimental_singleton. The documentation promises increased performance over 10x for @streamlit.experimental_memo when compared to @streamlit.cache. Unlike @streamlit.cache, it returns items by value, not reference, leveraging Python's pickle functionality.

Dash Dash follows a different philosophy than Streamlit. While Streamlit reruns the whole application upon any change, Dash functions by using callbacks. A callback is a decorator which maps UI elements to inputs and outputs of individual functions. It makes the application snappy and reduces loading times noticeably.

Breaking the application into several smaller units aids perfomance. Unfortunately, it makes sharing information between each of the units more difficult. Information sharing is necessary for the graphs are connected. Fortunately, Dash offers a simple solution in a form of a Dash component Store, making the issue easy to solve.

Dash also doesn't come with any formatting, so the application looks bare and unpolished out of the box.

Chapter 7

Results

XXX

Chapter 8

Concluding Remarks

XXX

Bibliography

Adan, A., G. Alizada, Y. Kiraz, Y. Baran, & A. Nalbant (2017): "Flow cytometry: basic principles and applications." *Critical reviews in biotechnology* **37(2)**: pp. 163–176.

Ashcroft, R. G. & P. A. Lopez (2000): "Commercial high speed machines open new opportunities in high throughput flow cytometry (htfc)." *Journal of immunological methods* **243(1-2)**: pp. 13–24.

Bassi, S. (2016): *Python for bioinformatics*. Chapman and Hall/CRC.

Bigos, M., N. Baumgarth, G. C. Jager, O. C. Herman, T. Nozaki, R. T. Stovel, D. R. Parks, & L. A. Herzenberg (1999): "Nine color eleven parameter immunophenotyping using three laser flow cytometry." *Cytometry: The Journal of the International Society for Analytical Cytology* **36(1)**: pp. 36–45.

Black, C. B., T. D. Duensing, L. S. Trinkle, & R. T. Dunlay (2011): "Cell-based screening using high-throughput flow cytometry." *Assay and drug development technologies* **9(1)**: pp. 13–20.

Bocchieri, E. & B.-W. Mak (2001): "Subspace distribution clustering hidden markov model." *IEEE transactions on Speech and Audio Processing* **9(3)**: pp. 264–275.

Campello, R. J., P. Kröger, J. Sander, & A. Zimek (2020): "Density-based clustering." *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* **10(2)**: p. e1343.

Nebe-von Caron, G., P. Stephens, C. Hewitt, J. Powell, & R. Badley (2000): "Analysis of bacterial function by multi-colour fluorescence flow cytometry and single cell sorting." *Journal of microbiological methods* **42(1)**: pp. 97–114.

De Biasi, S., L. Gibellini, E. Bianchini, M. Nasi, M. Pinti, S. Salvioli, & A. Cossarizza (2016): "Quantification of mitochondrial reactive oxygen species in living cells by using multi-laser polychromatic flow cytometry." *Cytometry Part A* **89(12)**: pp. 1106–1110.

Don, M. (2003): "The coulter principle: foundation of an industry." *JALA: Journal of the Association for Laboratory Automation* **8(6)**: pp. 72–81.

Dyer, R. & J. Chauhan (2022): "An exploratory study on the predominant programming paradigms in python code." In "Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering," pp. 684–695.

Ellyard, J. I., R. Tunningley, A. M. Lorenzo, S. H. Jiang, A. Cook, R. Chand, D. Talaulikar, A.-M. Hatch, A. Wilson, C. G. Vinuesa *et al.* (2019): "Non-parametric heat map representation of flow cytometry data: Identifying cellular changes associated with genetic immunodeficiency disorders." *Frontiers in immunology* **10**: p. 2134.

Finak, G., A. Bashashati, R. Brinkman, & R. Gottardo (2009): "Merging mixture components for cell population identification in flow cytometry." *Advances in bioinformatics* **2009**.

Fischer, B., V. Roth, & J. Buhmann (2003): "Clustering with the connectivity kernel." *Advances in neural information processing systems* **16**.

Fišer, K., T. Sieger, A. Schumich, B. Wood, J. Irving, E. Mejstříková, & M. N. Dworzak (2012): "Detection and monitoring of normal and leukemic cell populations with hierarchical clustering of flow cytometry data." *Cytometry Part A* **81(1)**: pp. 25–34.

Geng, X., Z. Wei, X. Yao *et al.* (2011): "Genetic variation of medicago sativa callus revealed by the ploidy analyser." *Acta Prataculturae Sinica* **20(3)**: pp. 156–161.

Gentleman, R. (2008): *R programming for bioinformatics*. Chapman and Hall/CRC.

Giorgi, F. M., C. Ceraolo, & D. Mercatelli (2022): "The r language: An engine for bioinformatics and data science." *Life* **12(5)**: p. 648.

Gong, Y., N. Fan, X. Yang, B. Peng, & H. Jiang (2019): "New advances in microfluidic flow cytometry." *Electrophoresis* **40(8)**: pp. 1212–1229.

Graham, M. D. (2013): "The coulter principle: imaginary origins." *Cytometry* **83(12)**: p. 1057.

Gratama, J. W., J.-L. D'hautcourt, F. Mandy, G. Rothe, D. Barnett, G. Janossy, S. Papa, G. Schmitz, R. Lenkei, E. W. G. on Clinical Cell Analysis *et al.* (1998): "Flow cytometric quantitation of immunofluorescence intensity: problems and perspectives." *Cytometry* **33(2)**: pp. 166–178.

Hall, S. E. & W. F. Rosse (1996): "The use of monoclonal antibodies and flow cytometry in the diagnosis of paroxysmal nocturnal hemoglobinuria." .

Herzenberg, L. A., J. Tung, W. A. Moore, L. A. Herzenberg, & D. R. Parks (2006): "Interpreting flow cytometry data: a guide for the perplexed." *Nature immunology* **7(7)**: pp. 681–685.

Hsu, C.-C., C.-L. Chen, & Y.-W. Su (2007): "Hierarchical clustering of mixed data based on distance hierarchy." *Information Sciences* **177(20)**: pp. 4474–4492.

Jacob, Y. & V. Pierret (1998): "Pollen size and ploidy level in the genus rosa." In "XIX International Symposium on Improvement of Ornamental Plants 508," pp. 289–292.

Jiménez, E., S. Contreras, N. Padilla, I. Zehavi, C. M. Baugh, & V. Gonzalez-Perez (2019): "Extensions to the halo occupation distribution model for more accurate clustering predictions." *Monthly Notices of the Royal Astronomical Society* **490(3)**: pp. 3532–3544.

Kalina, T., K. Lundsten, & P. Engel (2020): "Relevance of antibody validation for flow cytometry." *Cytometry Part A* **97(2)**: pp. 126–136.

Kamentsky, L. A. & L. D. Kamentsky (1991): "Microscope-based multiparameter laser scanning cytometer yielding data comparable to flow cytometry data." *Cytometry: The Journal of the International Society for Analytical Cytology* **12(5)**: pp. 381–387.

Key, M. (2012): "A tutorial in displaying mass spectrometry-based proteomic data using heat maps." *BMC bioinformatics* **13(16)**: pp. 1–13.

Le Meur, N., A. Rossini, M. Gasparetto, C. Smith, R. R. Brinkman, & R. Gentleman (2007): "Data quality assessment of ungated flow cytometry data in high throughput experiments." *Cytometry Part A: The Journal of the International Society for Analytical Cytology* **71(6)**: pp. 393–403.

Lo, K., R. R. Brinkman, & R. Gottardo (2008): "Automated gating of flow cytometry data via robust model-based clustering." *Cytometry Part A: the journal of the International Society for Analytical Cytology* **73(4)**: pp. 321–332.

Morissette, L. & S. Chartier (2013): "The k-means clustering technique: General considerations and implementation in mathematica." *Tutorials in Quantitative Methods for Psychology* **9(1)**: pp. 15–24.

Murtagh, F. & P. Contreras (2017): "Algorithms for hierarchical clustering: an overview, ii." *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* **7(6)**: p. e1219.

Nolan, J. P. (2015): "Flow cytometry of extracellular vesicles: potential, pitfalls, and prospects." *Current protocols in cytometry* **73(1)**: pp. 13–14.

Nolan, J. P. & L. A. Sklar (1998): "The emergence of flow cytometry for sensitive, real-time measurements of molecular interactions." *Nature biotechnology* **16(7)**: pp. 633–638.

Overton, W. R. (1988): "Modified histogram subtraction technique for analysis of flow cytometry data." *Cytometry: The Journal of the International Society for Analytical Cytology* **9(6)**: pp. 619–626.

Perez, O. D. & G. P. Nolan (2002): "Simultaneous measurement of multiple active kinase states using polychromatic flow cytometry." *Nature biotechnology* **20(2)**: pp. 155–162.

Picot, J., C. L. Guerin, C. Le Van Kim, & C. M. Boulanger (2012): "Flow cytometry: retrospective, fundamentals and recent instrumentation." *Cytotechnology* **64(2)**: pp. 109–130.

Pirone, D., M. Mugnano, P. Memmolo, F. Merola, G. C. Lama, R. Castaldo, L. Miccio, V. Bianco, S. Grilli, & P. Ferraro (2021): "Three-dimensional quantitative intracellular visualization of graphene oxide nanoparticles by tomographic flow cytometry." *Nano Letters* **21(14)**: pp. 5958–5966.

Räuber, S., M. Heming, J. Repple, T. Ruland, R. Kuelby, A. Schulte-Mecklenbeck, C. C. Gross, V. Arolt, B. Baune, T. Hahn *et al.* (2021): "Cerebrospinal fluid flow cytometry distinguishes psychosis spectrum disorders from differential diagnoses." *Molecular psychiatry* **26(12)**: pp. 7661–7670.

Robinson, J. P. (2004): "Flow cytometry." *Encyclopedia of biomaterials and biomedical engineering* **3**: pp. 630–642.

Roederer, M. & R. R. Hardy (2001): "Frequency difference gating: a multivariate method for identifying subsets that differ between samples." *Cytometry: The Journal of the International Society for Analytical Cytology* **45(1)**: pp. 56–64.

Rogers, W. T., A. R. Moser, H. A. Holyst, A. Bantly, E. R. Mohler III, G. Scangas, & J. S. Moore (2008): "Cytometric fingerprinting: quantitative characterization of multivariate distributions." *Cytometry Part A: the journal of the International Society for Analytical Cytology* **73(5)**: pp. 430–441.

Sachs, K., O. Perez, D. Pe'er, D. A. Lauffenburger, & G. P. Nolan (2005): "Causal protein-signaling networks derived from multiparameter single-cell data." *Science* **308(5721)**: pp. 523–529.

Schmitz, F., J. Glas, R. Neutze, & K. Hedfalk (2021): "A bimolecular fluorescence complementation flow cytometry screen for membrane protein interactions." *Scientific Reports* **11(1)**: pp. 1–9.

Shapiro, H. M. & W. G. Telford (2018): "Lasers for flow cytometry: current and future trends." *Current protocols in cytometry* **83(1)**: pp. 1–9.

Sieger, T., C. B. Hurley, K. Fišer, & C. Beleites (2017): "Interactive dendrograms: the r packages idendro and idendr0." *Journal of Statistical Software* **76**: pp. 1–22.

Srinath, K. (2017): "Python–the fastest growing programming language." *International Research Journal of Engineering and Technology* **4(12)**: pp. 354–357.

Steen, H. B. (2000): "Flow cytometry of bacteria: glimpses from the past with a view to the future." *Journal of microbiological methods* **42(1)**: pp. 65–74.

Sun, Z., G. Fox, W. Gu, & Z. Li (2014): "A parallel clustering method combined information bottleneck theory and centroid-based clustering." *The Journal of Supercomputing* **69(1)**: pp. 452–467.

Tratt, L. (2009): "Dynamically typed languages." *Advances in Computers* **77**: pp. 149–184.

Wilkerson, M. J. (2012): "Principles and applications of flow cytometry and cell sorting in companion animal medicine." *Veterinary Clinics: Small Animal Practice* **42(1)**: pp. 53–71.

Wilkins, M. F., S. A. Hardy, L. Boddy, & C. W. Morris (2001): "Comparison of five clustering algorithms to classify phytoplankton from flow cytometry data." *Cytometry: The Journal of the International Society for Analytical Cytology* **44(3)**: pp. 210–217.

Wood, B. (2006): "9-color and 10-color flow cytometry in the clinical laboratory." *Archives of pathology & laboratory medicine* **130(5)**: pp. 680–690.

Wrońska, A. K., A. Kaczmarek, J. Sobich, S. Grzelak, & M. I. Boguś (2022): "Intracellular cytokine detection based on flow cytometry in hemocytes from galleria mellonella larvae: A new protocol." *PloS one* **17(9)**: p. e0274120.

Young, I. T. (1977): "Proof without prejudice: use of the kolmogorov-smirnov test for the analysis of histograms from flow systems and other sources." *Journal of Histochemistry & Cytochemistry* **25(7)**: pp. 935–941.

Zamir, E., B. Geiger, N. Cohen, Z. Kam, & B.-Z. Katz (2005): "Resolving and classifying haematopoietic bone-marrow cell populations by multidimensional analysis of flow-cytometry data." *British journal of haematology* **129(3)**: pp. 420–431.

Zeng, Q. T., J. P. Pratt, J. Pak, D. Ravnic, H. Huss, & S. J. Mentzer (2007): "Feature-guided clustering of multi-dimensional flow cytometry datasets." *Journal of Biomedical Informatics* **40(3)**: pp. 325–331.

Zhang, J.-E., J.-H. Liu, & X.-X. Deng (2003): "Genetic variation of citrus calli revealed by the ploidy analyser." *Yi Chuan xue bao= Acta Genetica Sinica* **30(2)**: pp. 169–174.

Zhao, S., Y. Guo, Q. Sheng, & Y. Shyr (2014): "Advanced heat map and clustering analysis using heatmap3." *BioMed research international* **2014**.

Appendix A

Appendix One

A.1 Derivation of Desired Sample Size

XXX

Appendix B

R Source Codes

B.1 R Source Code for Dataset and Result Generation

XXX