Charles University in Prague
Faculty of Science

MASTER THESIS

Bc. et Bc. Nicole Aemilia Urban

Interactive Clustering Approaches in Single-cell Cytometry

Department of Cell Biology

Supervisor of the master thesis:	Mgr. Adam Šmelko
Study programme:	Bioinformatics
Study branch:	Bioinformatics

Prague 2024

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 interested in. I am also grateful for advice and encouragement I have received from them while working on the thesis.

Title: Interactive Clustering Approaches in Single-cell Cytometry

Author: Bc. et Bc. Nicole Aemilia Urban

Institute: Department of Cell Biology

Supervisor: Mgr. Adam Šmelko, Department of Distributed and Dependable Systems

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 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 real 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.

Keywords: flow cytometry clustering user interface data visualization

Contents

Introduction

Flow cytometry is a technique widely used in cell population analysis, particularly in clinical fields such as immunology or hepatology. It facilitates cell identification in various biological samples and as such can be useful in both research and clinical settings. The technique is based on a relatively simple principle: it measures fluorescence intensity when a laser beam travels through the sample. With flow cytometry, thousands of cells can be processed in a second. Although high processing speed is desirable, it can lead to processing challenges, as the vast amount of data that is collected has to be processed and analyzed.

The challenge of identifying cells has been traditionally solved by manual gating. Manual gating is a process in which the user needs to specify a region on plotted data - this can often be a scatter plot depicting two features - and draw boundaries around it. Everything that is a part of that region is then classified to be a cell of that particular type. Although manual gating requires manual labour and huge time investment, the process is still used today. But the identification of cell populations can also be formulated as a clustering problem. Hierarchical clustering combines the advantages of both supervised and unsupervised approaches as it automatically forms the clusters based on chosen metric, but a human remains responsible for cutting the dendrogram and choosing the proper number of clusters. The issue with hierarchical clustering is that the complexity of computation rises super-linearly with the amount of data. Furthermore, some variants of hierarchical clustering compute dissimilarity matrix, imposing quadratic space complexity. This further hinders its practical usability, especially for more complex datasets.

Recently, there have been many works on improving the performance of hierarchical clustering in various ways. As one example, Smelko et al. [1] took advantage of parallel accelerators by implementing GPU-accelerated method. This work demonstrated order of magnitude speedup and improved the scalability of the methods, which opens doors to processing bigger and more complex datasets.

The aim of this thesis is to bridge the gap between methodological advancements and domain experts, who often come from environments with little programing background. The thesis demonstrates how user-supervised hierarchical clustering outperforms manual gating in terms of convenience, speed, and accuracy that can be achieved even on large datasets and with no requirement of programming experience from the user.

As a practical result, this thesis presents Ash, a software that empowers domain experts to conveniently analyze their data and use hierarchical clustering to assign data points to clusters via graphical user interface. The intended workflow is that the user imports their data, inspects various visualizations, assigns data points to clusters by cutting dendrogram branches, and finally exports the newly formed cluster assignment to *.csv* file. The respective data can come from `gMHC` [1] or any other widely used clustering tool such as the R `hclust` package.

Ash draws inspiration from iDendro [2] R package and adds functionality on top of it. There are three main advantages Ash offers:

- a more granular method for splitting clusters,
- usable scalability on larger data,
- no programming knowledge requirement.

The custom method for user-supervised cluster assignment and the conversion tool between Python and R dendrogam formats are published as a standalone Python package on PyPI.

Further in this thesis, the practical usability of Ash is tested by identifying incorrectly labeled eosionphils in Samusik dataset [3]. Corrected labels are also published along with this thesis. The aim is to demonstrate that with such tool as Ash, domain experts can be empowered to perform analysis that was previously reserved for data scientists.

The thesis is structured into four chapters. The first chapter focuses on flow cytometry, describes how it works and what data are generated during the process, and why user-supervised hierarchical clustering is suitable for this problem. Furthermore, possible issues with manual gating are discussed and existing solutions are reviewed. In the second chapter it is explored where Ash fits into the landscape of other interactive clustering tools. Architectural decisions behind Ash are briefly described, what technology was used and why is explained. Three ways to use Ash are described, one of them being the available web application, the second one being the local installation, and the third one being a standalone Python package. The third chapter showcases the application of Ash on real life data and presents the collected evidence for the mislabeling of eosinophils in Samusik dataset [3]. The fourth and final chapter concludes the thesis and suggests possible future work.

Chapter 1

Introduction to Flow Cytometry Data Analysis

In this chapter, the concept of flow cytometry is explained. This chapter should help the reader understand how flow cytometry works and get familiar with flow cytometry related terminology. Being familiar with the process can help the reader better understand the information that is produced and analyzed in this thesis. Several methods used for flow cytometry data analysis are introduced, with the focus mainly on manual gating and hierarchical clustering. In the final section, an overview of data visualization techniques is given.

1.1 Principles of Flow Cytometry

Flow cytometry is a laboratory technique used in both qualitative and quantitative cell analysis. It can be used for sorting cells based on their physical and chemical properties. In research, flow cytometry plays a crucial role in providing insights into cellular processes, functions, and interactions. For instance, it is used to study immune responses, cell cycle progression, and apoptosis. In the industry, flow cytometry is indispensable for diagnosing and monitoring a wide range of diseases, including cancers such as leukemia and lymphoma, immune disorders, and infections. By analyzing cell surface markers and intracellular molecules, clinicians can identify abnormal cells, assess disease progression, and tailor treatments to individual patients. A more in depth review can be read in McKinnon [4].

In order to understand flow cytometry, it is important to understand the basic three principles it builds upon:

- laser beams can be used to identify the properties of the cells passing through the beam,

- the cells have to be separated so that they flow through the flow cytometer one by one,

- and antibodies with fluorescent markers can identify cells of interest.

The following paragraphs elaborate on these principles.

Light scattering is a process in which particles deviate from their trajectory when they encounter an obstacle. In case of flow cytometry, the obstacle is a cell passing through the beam. When detectors are put on the opposite side of the passing cell, they can measure the light that was scattered. A detector directly opposite to the laser beam is called forward scatter detector and it can be used to measure the size of the passing cell. The other detectors are placed at 90 degrees to the laser beam and are called side scatter detectors. They measure the granularity of the cell, such as its organelles or specific proteins the cell expresses [5].

To ensure that only one cell at a time passes through the laser beam, sheath fluid is used to circulate through the cytometer. The fluid ensures that constant flow is maintained. The sample is injected by droplets under different pressure than the sheath fluid and the separation is achieved by passing the sample through narrow nozzle that vibrates. Flow cytometer cell schema can be seen in Figure 1.1.

Cytometers do not have to measure only the size and the granularity of cells. They can also measure the presence of specific proteins. This is achieved by using antibodies that bind to the protein of interest. They can be labeled either directly with a fluorescent marker or indirectly by using a secondary antibody that is labeled. Fluorophore molecules are commonly used for labeling purposes. They are a fluorescent chemical compound that can re-emit light of certain wavelength upon light excitation [6].

If a device has detectors that are able to detect these wavelengths, it can measure the presence of the specific protein. The more protein is present, the more light is emitted and the more light is detected. Commonly, multiple detectors are contained in a single cytometer [7]. Emitted light can be directed by mirrors and multiple antibodies can bind to same cell [8]. That allows magnitudes of different measurements per sample, although technically it is limited by the number of detectors.

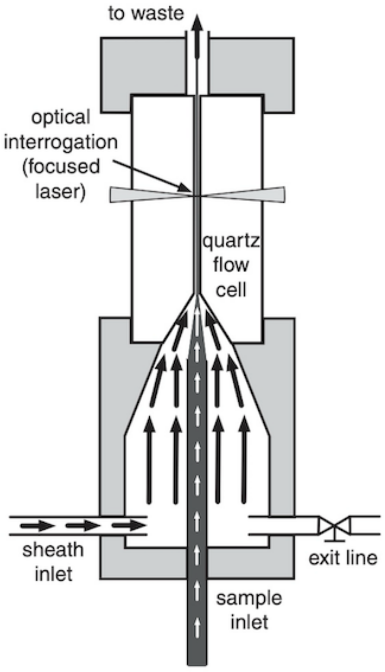


Figure 1.1 Schema of a flow cytometer cell used for collecting information [5].

1.2 Flow Cytometry Output Data

This section discusses the way flow cytometry output data are stored and what type of data is gathered. The output of flow cytometry is a complex dataset that provides detailed information about individual cells in the sample. The output data provides an entry for each cell and parameters that were measured for the cell, such as size or the amount of specific proteins.

One of the main strengths of flow cytometry is its cost-to-output ratio. Thousands of particles can be evaluated per second. Usual throughput of a cytometer ranges between 100-1000 cells per second [9]. That unavoidably leads to large datasets. Additionally, recent advancements, such as the use of fluorescent dyes, have significantly increased dataset output size [10]. Up to 10^5 of rows could be expected in a single dataset in resulting from one milliliter of the sample [9].

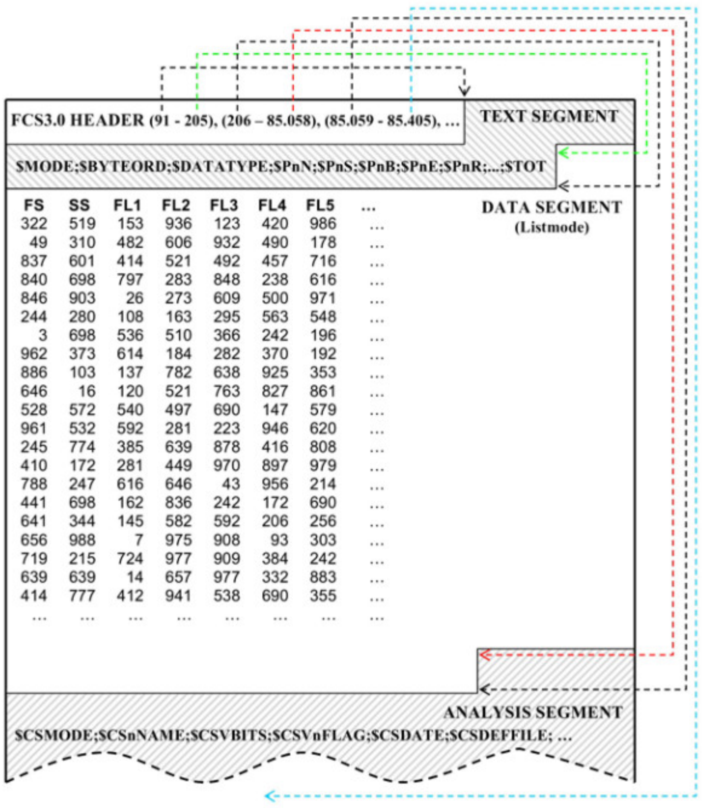
In addition to fluorescence, flow cytometry records frontal scatter and side scatter. However, the number of dimensions or columns in columnar format is limited by hardware and fluctuates around couple of tens of dimensions, based on the type of the respective machine.

1.2.1 Cytometry Data Format

The output data is stored in the flow cytometry standard format with *.fcs* extension and was developed by the International Society for Advancement of Cytometry [11].

Figure 1.2 shows the structure of an *.fcs* file structure. Notably, the HEADER contains information about the version used ("FCS3.0"). The arguably most important part of the file is the DATA segment, which encodes the measurements for each particle or cell in the sample.

From programming perspective, there are several ways how *.fcs* could be read. Most notably, Python as well as R both contain multiple packages that allow that. Nevertheless, reading binary data is supported by virtually all programming languages, so the format itself is not a limiting factor.


FCS3.0 HEADER (91 - 205), (206 – 85.058), (85.059 - 85.405), …
TEXT SEGMENT
$MODE;$BYTEORD;$DATATYPE;$PnN;$PnS;$PnB;$PnE;$PnR;...;$TOT
DATA SEGMENT
(Listmode)
ANALYSIS SEGMENT
$CSMODE;$CSnNAME;$CSVBITS;$CSVnFLAG;$CSDATE;$CSDEFFILE; ...

FS	SS	FL1	FL2	FL3	FL4	FL5	…
322	519	153	936	123	420	986	…
49	310	482	606	932	490	178	…
837	601	414	521	492	457	716	…
840	698	797	283	848	238	616	…
846	903	26	273	609	500	971	…
244	280	108	163	295	563	548	…
3	698	536	510	366	242	196	…
962	373	614	184	282	370	192	…
886	103	137	782	638	925	353	…
646	16	120	521	763	827	861	…
528	572	540	497	690	147	579	…
961	532	592	281	223	946	620	…
245	774	385	639	878	416	808	…
410	172	281	449	970	897	979	…
788	247	616	646	43	956	214	…
441	698	162	836	242	172	690	…
641	344	145	582	592	206	256	…
656	988	7	975	908	93	303	…
719	215	724	977	909	384	242	…
639	639	14	657	977	332	883	…
414	777	412	941	538	690	355	…
…	…	…	…	…	…	…	…

Figure 1.2 The structure of *.fcs* file from Drakos et al. [12].

1.3 Flow Cytometry Data Analysis

This section describes how manual gating can be used on flow cytometry data. Additionally, clustering is introduced, as it is the approach this thesis build on. The section also discusses some of the limitations of manual analysis techniques and why there is a room for improvement in this kind of analysis.

1.3.1 Manual Analysis Techniques

Despite technological advancements, it is still common to manually group cells into clusters in a process called manual gating. However, as new technology enables more and more parameters to be collected, this process becomes tedious and time-consuming.

Manual gating is a technique used to create subsets of cells based on a 2D projection of flow cytometers output. The user can manually define regions called "gates". Gates can have various dimensions and shapes. An example of gated data can be seen in Figure 1.3. However, the complexity of manual gating grows with each marker added to the analysis, as every combination needs to be analyzed. For example, with 20 markers there are $\binom{20}{2} = 190$ possible 2D plots.

According to Maecker et al. [13], analysis, particularly gating, is a significant source of variability in the results. Maecker et al. [14] later suggest that gating can be the largest source of variability. Additionally, every manual gating approach relies on the researcher's prior knowledge, thus introducing a bias toward "expected" results [15]. Parts of the variability might come from other sources, such as hardware.

Due to the apparent simplicity of manual gating, many researchers have attempted to automate the gating process. Automation offers advantages such as human error prevention and speed increase. However, automated gating becomes challenging with increasing non-convex cell populations, which have complex shapes that curve in various directions, or other multidimensional shapes. Automated systems can struggle with elliptical shapes that are often produced by flow cytometry [16].

1.3.2 Clustering Methods

Clustering is an essential method in data analysis and machine learning, used to group a set of objects into clusters based on their similarities. It is a form of unsupervised learning, meaning it does not rely on pre-labeled data. Instead, clustering algorithms discover the inherent structure in the data by organizing objects that are more similar to each other, according to chosen metric, into the same cluster, while placing dissimilar objects into different clusters. The primary

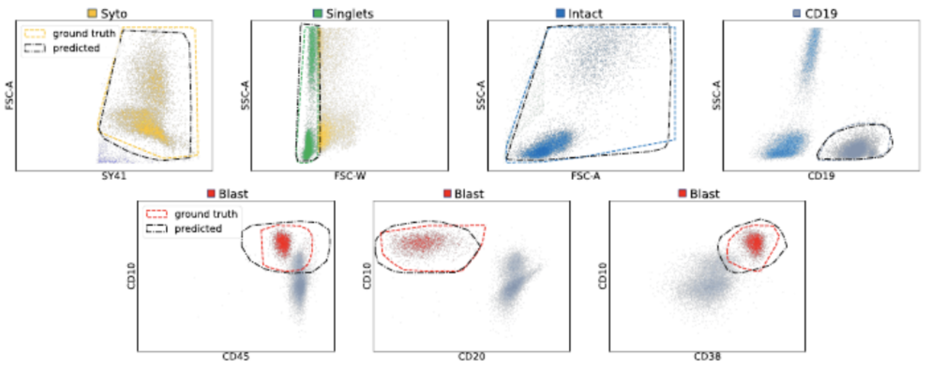


Figure 1.3 Example gated data from Kowarsch et al. [17].

goal of clustering is to ensure that objects within the same cluster have high degree of similarity, while objects in different clusters are as distinct as possible. This is achieved through various clustering methods, each with its own approach to defining and identifying clusters. The choice of a clustering method depends on the nature of the data and the specific requirements of the analysis. Clustering has a wide range of applications across different fields: in marketing, it can be used for customer segmentation; in biology, for classifying species or genes; in image processing, for object recognition; and in social network analysis, for community detection.

There are many possible approaches to clustering such as centroid-based clustering, density-based clustering, distribution-based clustering, and hierarchical clustering [18].

The following paragraphs explore hierarchical clustering in detail, as it is the main approach relevant to this thesis.

Hierarchical Clustering

Hierarchical clustering, also known as connectivity-based clustering, is an algorithm that groups objects based on their similarity. It provides insights into the data structure by allowing associations between subclusters. It differs from other clustering methods by not requiring the number of clusters to be specified and instead creates a tree-like structure that shows how clusters would be formed for different similarity thresholds.

There are two main approaches to hierarchical clustering: bottom up, which is also known as agglomerative, and top down, which is also known as divisive. In the bottom-up approach, the algorithm starts with individual data points and iteratively merges the closest elements into clusters. The newly merged points become a new cluster that is referred to as a “node”, and nodes are further merged with the rest of the data points and nodes until one single cluster is formed. In comparison, the top-down divisive approach begins with all data points in a single cluster and recursively splits it into smaller clusters. Only the agglomerative approach is discussed in this thesis.

One common way to visually represent hierarchical clustering is through dendrograms, which will be described later in the thesis.

Distance Metrics in Hierarchical Clustering

The choice of a distance metric influences the outcome of clustering algorithms. In this section, the foundation for understanding these metrics is established. Clustering algorithms are usually agnostic to distance metrics, which means that the user can provide their own metric. No single metric is universally better, as the choice depends on the type of data and often also on computational speed and intuition. The metrics capture the notion of closeness of two data points.

According to Chen et al. [19], distance metric is a real valued function $d(x, y)$ defined on $S \times S$, where S is a set of points, that can be compared ($x, y \in S$), provided that it satisfies following conditions:

1. $d(x, y) \geq 0$ (non-negativity)
2. $d(x, y) = d(y, x)$ (symmetry)
3. $d(x, z) \leq d(x, y) + d(y, z)$ (triangle inequality)
4. $d(x, y) = 0$ if and only if $x = y$ (identity of indiscernibles)

All four of these properties originate from the notion of distance in the real world. Non-negativity ensures distance between objects has to be non-negative and it is zero if and only if these objects actually refer to the same point (identity of indiscernibles). Symmetry guarantees that if the measurement is taken from point x to point y the same measurements are obtained as from y to x. Lastly, triangular inequality captures the fact that the direct path between x and z must not be longer than the most probable indirect path that also goes though y.

In Kumar et al. [20], it is shown how the choice of metric influences the resulting performance of the clustering algorithm in terms of accuracy, inter-cluster and intra-cluster distances. The paper uses ten datasets (some artificial, some real) and concludes there is no single winner. Comparison is also attempted by Singh et al. [21]. Despite conclusive results suggesting superior performance of Euclidean distance, their applicability to this thesis remains questionable as the authors uses different algorithm (k-means clustering) and only a single dataset. The default distance metric is usually Euclidean, although in flow cytometry context, the use of Mahalanobis distance might be preferred [22].

Selected commonly used metrics The most commonly used distance metric is called Euclidean distance. It is represented by the following formula:

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(y_i - x_i)^2}$$

It extends the Pythagorean theorem to n-dimensional spaces, therefore is often the most intuitive choice. The potential issue with Euclidean distance is that if variables have different scales, some of them will affect the resulting distance more than others. That can be solved by Mahalanobis distance. Mahalanobis distance between points x and y with respect to a covariance matrix S is defined as follows:

$$d(x,y) = \sqrt{(\mathbf{x} - \mathbf{y})^T S^{-1} (\mathbf{x} - \mathbf{y})}$$

Notice the vector notation, as opposed to the notation of Euclidean distance. S is covariance matrix of target variables and it has to be either known or estimated. Put simply, the introduction of the inverted covariance matrix basically transforms coordinate system of our variables, so that they are not correlated anymore and variance of each equal zero. Then, Mahalanobis distance equals Euclidean distance on the transformed data.

There are many more commonly used metrics, such as Minkowski distance. Minkowski distance is a name used for an entire family of metrics parametrized by p and defined as:

$$d(x,y) = (\sum_{i=1}^{n} |y_i - x_i|^p)^{1/p}$$

Notice that for $p = 2$, Minkowski distance equals Euclidean distance. Another common parameter choices are $p = 1$ and limiting case when p approaches infinity. Resulting metrics are called Manhattan distance and Chebyshev distance respectively. As p approaches infinity: $d(x,y) = \max_{i=1}^{n} |y_i - x_i|$.

Contrary to popular belief, cosine distance is not a distance metric as it does not satisfy identity of indiscernibles. It is defined as:

$$d(x,y) = 1 - \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} y_i^2}}$$

Cosine distance does not solve the curse of dimensionality. In layman's terms, the curse of dimensionality refers to distance metrics losing their usefulness as dimensionality of the data approaches infinity. Cosine similarity is typically used on high dimensional data in text analysis, where text embeddings can have up to hundreds of thousands dimensions, but such embeddings often contain only few non-zero numbers. The best uses for cosine similarity are very sparse, discrete domains such as the aforementioned text analysis. Therefore, it is not suitable for flow cytometry.

Clustering in Flow Cytometry

As discussed in Section 1.3.1, flow cytometry data are traditionally analyzed by manual gating. However, manual gating has limitations, such as requiring deep knowledge of cell biology for data interpretation. It is highly subjective, time-consuming, and with increasing data size, the complexity requirements increase. Analyzing flow cytometry data can be defined as a clustering problem, and clustering approaches can be used to facilitate data analysis. Especially with recent advancements, hierarchical clustering can be fast even on larger datasets, and using the right distance metric for the problem, such as the Mahalanobis distance, can also increase the effectiveness of clustering [1].

Hierarchical clustering outputs are tree-like structures named dendrograms, which allow the researchers to explore relationships between individual cells and cell populations. By utilizing hierarchical clustering, it becomes feasible to align cell subsets across different samples, which facilitates direct comparisons. Furthermore, by leveraging shared information across multiple samples, sensitivity can be enhanced for detecting low-frequency cell subsets [23].

However, clustering data can bring some pitfalls. In biological context, it can be difficult to discern whether the clustering produced biologically relevant results as there can be nuances in the data that cannot be accessed via unsupervised learning algorithms. It is important to know about the common pitfalls and how to avoid them, as proposed by Ronan et al. [24].

In the next section, various visualization techniques which can help users indicate clustering errors are presented.

1.4 Data Visualization in Flow Cytometry

It his section, data visualization techniques are reviewed. Additionally, their meaning in the context of flow cytometry and what can be learned from each one of them is discussed.

Data visualization helps translating data and information into a graphical representation. Visual representation can help researchers understand their data better. Especially large datasets can be hard to navigate and understand, and it is often important to make informed decisions. Data visualization is useful in many fields including network security [25], improvement of services [26], money laundering and other crime detection [27, 28]. A review of how data visualization can be useful in the medical field can be found in Park et al. [29] and Aung et al. [30].

Graphical representation can make it easier to identify patterns, trends, and outliers. Relationships between datapoints can be represented by various plots depending on whether it is desirable to represent relationships or connections in a form of a network. It is also believed that data visualization is closely related to human creativity [31, 32].

Data visualization can be either static or interactive. Static data visualization displays data in a fixed format. The advantage of an interactive solution is that it allows the user to explore the data with various filters, highlighting possible subsets of the collected data. A proof of concept of how switching plots from static to interactive can improve data exploration can be found in Weissgerber et al. [33].

In the following paragraphs, an overview of the most typically used visualization techniques is given.

Heatmap A heatmap is a 2D grid. In a heatmap, data is represented by mapping values to colors, making it easy to spot areas with similar marker levels. Another advantage of a heatmap is that the user can visualize large amounts of data compactly, because labeling values by color takes less space. An example of a heatmap is shown in Figure 1.4.

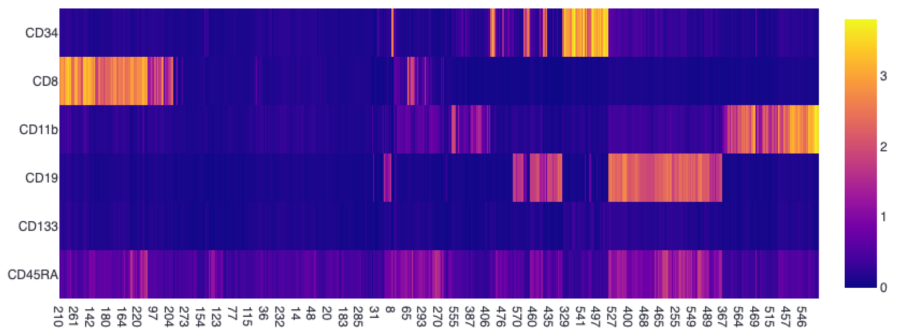


Figure 1.4 An example of a heatmap which shows protein markers on y-axis and individual samples on x-axis based on sampled data from Levine_32 dataset [34]. The continuous color-scale represents protein marker levels. Patterns can be spotted visually in such representations and can help with differentiating cell populations.

In flow cytometry context, users should focus on areas with similar colors, as they represent similar marker levels. It serves a purpose of identifying clusters based on marker values (similar shades) and as a visual confirmation that cells with same values are within same cluster.

Dendrogram A dendrogram, also known as a tree diagram, is a 2D data representation that is designed to reveal the hierarchical structure of data. The use of dendrograms is popular in biology, where it can be useful for taxonomy [35] and visualization of multiscale networks [36], or structural analysis [37]. An example dendrogram is shown in Figure 1.5.

The data points are called leaf nodes. Leaf nodes are linked together by branches, and branches are joined together in nodes. Clusters share the same upper nodes, and color can be used to highlight separate nodes, especially in dendrograms which show a lot of datapoints.

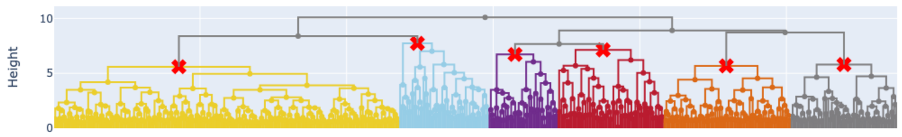


Figure 1.5 An example of a dendrogram. This dendrogram represents data from Levine et al. [34] and was plotted is Ash. Leaf nodes are individual cell samples. The hierarchical clustering algorithm took into account features that represent marker levels.

Scatter plot Scatter plots are a common statistical chart type that represent data using dots. In the plot, each dot is a representation of a data point. These plots are useful for visualizations of distribution in a dataset, especially for continuous, quantitative, univariate data. In manual gating, these plots are the place where user draws gates. Scatter plots are particularly useful for small data sets and they allow outliers detection. An scatter plot example can be seen in Figure 1.6.

Dimensionality reduction Dimensionality reduction techniques are commonly used to plot high-dimensional data in low-dimensional space, allowing for exploration of data structures that can be difficult to identify otherwise. While humans can comfortably plot and explore data in 2D and 3D spaces, higher dimensions are challenging and remain hard to plot and therefore explore visually. The aim of dimensionality reduction algorithms is to preserve the original structure of the data while making exploration possible. To overcome this challenge, well-known dimensionality reduction algorithms were developed such as principal component analysis (PCA), t-Distributed Stochastic Neighbor Embedding (t-SNE), or uniform manifold approximation and projection (UMAP).

A comparison of how different dimensionality reduction algorithms handle the same input data can be seen in Figure 1.7.

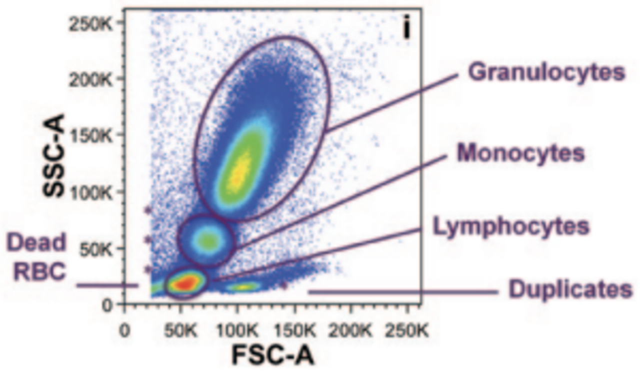


Figure 1.6 An example of a scatter plot as used in flow cytometry from Jahan-Tigh et al. [38]. Rare events and high-frequency area can be located easily.

Principal Component Analysis (PCA) PCA is an unsupervised algorithm based on linear combinations of original variables. It creates a new set which consists of of uncorrelated variables which capture variation in the original data. It is also applicable to large datasets. However, it can struggle with datasets that have non-linear structure or are non-normally distributed.

PCA is sensitive to scaling and works better on normalized data. It may also be difficult to interpret the meaning of the principal components as the components don't have intuitive meaning in terms of original variables.

An overview of using PCA in flow cytometry can be found in Lugli et al. [39].

t-Distributed Stochastic Neighbor Embedding (t-SNE) In contrast, t-SNE is a non-linear technique that can better handle data with underlying non-linear structure. It preserves pairwise distances between data points and can identify outliers, but it is computationally expensive and may require fine tuning. Computational expenses grow with the number of dimensions. Additionally, t-SNE is not deterministic and due to its non-parametric nature cannot be used as a classification method for previously unseen data points. A comparison of conventional and t-SNE guided gating can be found in Eshghi et al. [40].

Uniform Manifold Approximation and Projection (UMAP) UMAP is similar to t-SNE in many aspects. It aims to preserve local structure rather than global. UMAP can work well with outliers and can handle large datasets. It also scales well. However, UMAP may require fine tuning and is not guaranteed to converge to a local minimum. More about using UMAP in flow cytometry is reviewed in Stolarek et al. [41].

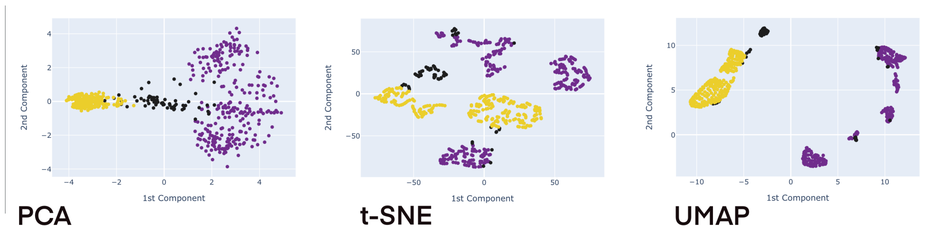


Figure 1.7 An example of PCA, t-SNE, and UMAP done on data from Levine et al. [34]. These plots were generated in Ash. The data was reduced from 32 dimensions to 2 dimensions.

Chapter 2

Implementation of Ash

The following chapter discusses the process of creating an interactive visualization tool named Ash. Firstly, currently available clustering tools for flow cytometry are reviewed. Secondly, selection of appropriate tools and frameworks is explored, and how to use available tools to create new solutions to explore flow cytometry data is discussed. Thirdly, the architectural design and challenges of the final solution are described. Finally, the user interface of Ash is described.

2.1 Overview of Interactive Clustering Tools for Flow Cytometry

This section briefly describes available tools for interactive clustering. For the purposes of this section, available tools are compared across four dimension. These are:

- Licensing model and respective cost
- Support for clustering methods
- Required programming knowledge
- Web-based or desktop-based interface

Ash is compared to FlowJo [42], FlowLogic [43], Cytobank [44], AUTOKLUS [45], iDendro [46], XCluSim [47], Geono-Cluster [48].

For the purpose of this thesis, a distinction is not made between the nuances of particular licenses. Instead, the focus lies on the price and whether the source code can be subjected to inspection. Arguably, the most popular tools are only available commercially. Among such solutions, FlowJo is the most established one. FlowJo is mainly focused on gating, although a clustering module is also available. According to the documentation [49], it offers k-means and k-nearest neighbours, but hierarchical clustering, that this thesis focuses on, is not available. The method that comes closest to the Ash approach is the FlowSOM plugin. The plugin supports clustering and uses dendrograms combined with heatmaps to display clusters similar to Ash. FlowLogic, a FlowJo alternative, is also commercial and does not offer the functionality to use or plot hierarchical clusters either. Both offer a vast amount of features and are available as desktop applications and do not require any programming knowledge. Contrary to them and similar to Ash, Cytobank offers a web-based interface instead of a desktop one and also provides some clustering methods such as FlowSOM and SPADE. But similar to FlowJo and Flowlogic it, is also not freely available. All of these tools support reading *.fcs* files in drag-and-drop fashion and are well regarded within the field for their versatility and reliability.

Among the freely available GUI tools are Geono-Cluster [48] , XCluSim [47], iDendro [2] and AUTOKLUS [50]. Geono-Cluster aims specifically to help domain experts without data science training to merge and split clusters via a GUI using a drag-and-drop interface. The intended workflow is that the user indicates an initial cluster assignment, and Geono-Cluster chooses the most appropriate method to achieve that assignment. Supported algorithms include k-means, DBScan, agglomerative clustering, and spectral clustering [48]. The implementation of algorithms is not exposed to the user. On the other hand, XCluSim focuses on comparing different clustering algorithms, not on the cluster assignment itself. It uses dendrograms as a tool to compare the similarity between different cluster assignments. iDendro is a tool that inspired Ash the most. iDendro enables the user to plot large dendrograms, which can be zoomed in, panned, and inspected interactively by selecting and coloring clusters anywhere in the dendrogram [2]. Ash strives to replicate that without requiring R language knowledge. AUTOKLUS only runs on Windows [50], supports k-means but is mostly outdated. The differences between the aforementioned tools are highlighted in Table 2.1.

This thesis and Ash are not limited to any single metric, as the expectations are that users provide their own data in the form of dendrogram, regardless of the distance metric used to construct it. The input format is specified further in the Section 1.2. Ash offers users the option to analyse their data from hierarchical clustering algorithm variation of their choice, and helps them see patterns in the data and manipulate cluster assignment of the individual data points.

2.2 Selected Technologies and Architecture

In this section, an overview of technologies that were considered is presented. Arguments for the Python, Plotly and the selected Dash stack are discussed.

Tool	**Licencing Model**	**Clustering Support**	**Programming Knowledge**	**Graphical Interface**
FlowJo	commercial	multiple	not required	desktop
FlowLogic	commercial	multiple	not required	desktop
Cytobank	commercial	multiple	not required	web
AUTOKLUS	free	k-means	not required	desktop
iDendro	free	user-supervised hierarchical	limited	desktop
XCluSim	free	compare only	not required	desktop
Geono-Cluster	free	multiple	not required	desktop
Scipy	free	multiple	required	desktop
hclust	free	hierarchical	required	desktop
Ash	free	user-supervised hierarchical	not required	web

Table 2.1 Comparison of visualisation tools.

Section 2.2.1 discusses the choice of Python as the programming language and its advantages compared to R. Section 2.2.3 reviews available plotting frameworks and provides a rationale for selecting Plotly. Section 2.2.2 discusses the choice of Dash and how it compares to Shiny, Streamlit and PyScript.

2.2.1 Programming Language

Ash is written in the programming language Python 3. Python was chosen as it is a widely popular language, suitable for data analysis and has a lot of tools to facilitate working with data. In Python's ecosystem, it is easy to distribute packages so other people could use them for their projects.

When open-sourcing a project, choosing a popular tool is important to make tool customization accessible to users who are interested. Popularity is even more important when available tools and libraries are considered. Compared to R, the community has been leaning towards Python recently [51], which resulted in more people who are familiar with Python and more tools that are available for Python.

The popularity has driven the decision to use Python for Ash instead of R, but R would be suitable as well. Other languages often lack the tools focused on data analysis, which Python and R both have.

2.2.2 Interactive Dashboarding Frameworks

This sections compares the three most popular interactive dashboard frameworks for Python: Dash, Streamlit, and Shiny. Dashboarding frameworks help create well-rounded dashboards, connecting individual elements together. In Ash, dashboarding serves as glue between individual visualizations and menus. Dash and Streamlit are Python only frameworks, compared to Shiny, which originates from the R ecosystem but has been available for Python since 2023. Each of them can be installed from Python packaging index and on high level, they all provide comparable functionality. At their core, they are all web building frameworks that set up a web server that listen to user requests (made through a web browser) and they change the content of the page accordingly. This approach enables straightforward deployment and sharing of the application that is further described in Section 2.3.3.

For the purposes of this thesis, they mainly differ in three areas: level of abstraction, backend architecture, and maturity. This thesis does not imply that any of the frameworks is superior but rather that these frameworks are suitable for different use cases. Overview of the differences can be inspected in the Table 2.2.

Level of Abstraction

All of the frameworks are considered to be high level because the user does not have to concern themselves with the underlying web server, nor with CSS, JavaScript, and other web technologies. That can be either an advantage or a disadvantage, depending on the use case. The amount of control over the outcome is traded off for the ease of use. Among the three frameworks, there are still differences in the level of abstraction, most notably between Streamlit and the other two.

Streamlit focuses on extreme simplicity and is well suited for developing workable prototypes as fast as possible. The main disadvantage of that approach is low level of control and customization. That is demonstrated the best on the fact that Streamlit lacks the ability to trigger an update of a specific part of the page, something which both Dash and Shiny can do.

The implication is that any time a change occurs anywhere in the application, Streamlit recalculates the entire page regardless of what has changed. Shiny and Dash use Python decorators to link an output element with UI elements which leads to updating only the necessary parts of the page. Updating the whole page is not an issue when all the calculations are fast, but one slow calculation suffices to make the user experience unpleasant. In case of repetitive long-running calculations, Streamlit offers caching mechanism that loads the precomputed results instead of recalculating them.

On the other hand, controlling what gets updates can easily lead to code that is hard to understand and maintain. The reasoning behind the code and debugging it gets harder, especially when one trigger initiates a chain of other triggers.

Back-end Architecture

In terms of back-end architecture, the most notable differences between the frameworks are used web protocols and statelessness or statefulness of the framework. Both of these dimensions are of concern for the scalability rather than for the local run. Statelessness is a server property that does not store any information regrading client's session, which allows seamless switching between multiple servers and helps distributing the load. This process is also known as scaling the application horizontally. That can be advantageous if the application is expected to be used by many users at the same time. However, scientific workloads are often CPU-bound, so the horizontal scalability on its own may not ensure the best performance. From the three frameworks, only Dash is stateless. But the importance of statelessness is dependent on the use case.

The frameworks also differ in the web protocol they use. Only Dash uses HTTP(S), while Streamlit and Shiny both use WebSockets. In case of WebSockets, the connection between the server and the client is kept open, which allows for more complex communication patterns, such as server periodically pushing updates to the client. That is well suited for use cases where data is changing in real time and the server does not wait for the client to request the data. However, the open connection is one additional obstacle for scaling as requests cannot be load balanced (responded by different servers) easily.

On the other hand, a lot of resources exist on HTTP(S) can be load balanced, but the user has to actively request the data.

Maturity

Maturity of the framework influences how often the framework is updated, how many outstanding issues it has, and how big the community is. Despite the fact that no universal metric for maturity exists, the differences in maturity between Dash and Streamlit are arguably negligible, but one can argue that Shiny is the least mature of them. That is mainly because Shiny for Python was released in 2023. On the other hand, it is a ported variant of the reputable Shiny for R, which was released in 2012. Other possible metric is the amount of stars on GitHub, which is summarized in the Table 2.2.

	Dash	**Shiny**	**Streamlit**
Abstraction Level	high	high	very high
Partial Updates	yes	yes	no
Web Protocol	HTTP(S)	WebSockets	WebSockets
Statefulness	Stateless	Stateful	Stateful
Maturity	high	medium	high
Stars on GitHub	21k [52]	1.1k [53]	33.6k [54]

Table 2.2 Dashboarding frameworks comparison

2.2.3 Plotting Frameworks

In this section, the most popular plotting Python frameworks are compared: Matplotlib, Seaborn, Plotly and Bokeh, and Altair. Plotting frameworks are essential for creating coherent data visualizations. They can help transform data into actionable insights. The chosen frameworks are compared on the *level of abstraction*, *interactivity*, and *maturity*. Same rationale as in the previous section applies:

- The higher the level of abstraction, the more control is traded for ease of use.
- No framework is inherently superior but rather more suitable for different use cases.

Level of Abstraction

Plotting frameworks cannot be evaluated in isolation because a lot of them are often just high level interfaces on top of another plotting library. Among the five frameworks, only Matplotlib and Bokeh are not built on top of something else. Seaborn is built on top of Matplotlib, and Plotly and Altair are built on top of D3.js, which is a JavaScript plotting library. Seaborn targets statistic plotting and Altair and Plotly focus on providing different interface. Frameworks are built on top of another one usually for increasing user convenience. That makes Plotly, Altair, and Seaborn all high level frameworks. Plotly offers even higher level of abstraction with Plotly Express, which is a wrapper around Plotly.

On the other hand, Matplotlib and Bokeh are both low level frameworks that allow very fine-grained control over what is plotted. Matplotlib is the oldest of the five, and it aspired to bring Matplot-like plotting to Python. It offers two separate interfaces but both are verbose and low level. Bokeh differentiates itself from Matplotlib by focusing more on interactivity but it is still a low level framework.

Interactivity

Interactivity was introduced in the Section 1.4 as a key feature of visualization tools. The frameworks that are compared in this chapter all apart from Seaborn offer some level of interactivity.

In case of Matplotlib, interactivity is limited as it was not a main focus of the library. That clarifies why no interactivity is present in Seaborn, as it is built on top of Matplotlib.

Altair and Plotly are built on top of D3.js, therefore they are both producing plots in web compatible formats that can utilize interactivity and can be easily shared. The advantage of that is that all interactivity happens in the browser, which does not require any additional installation from user. Neither Altair nor Plotly specializes on displaying large datasets and the difference between them is mainly in the philosophy of the library interface and integration. Plotly and Dash are both maintained by the same company, which leads to better integration between the two. Alternatively, using Altair in a Dash application is possible as well.

Bokeh's output can be also displayed in the browser, turned to dashboard and exported to HTML with the interactivity in mind.

Maturity

The most mature of the five is Matplotlib, but in terms of maturity all frameworks are mature enough to be used in production environments without hesitation. As a proxy variable to judge the maturity, the number of stars on GitHub is used, similarly to the previous section. See the Table 2.3 for the summary.

	Matplotlib	**Seaborn**	**Plotly**	**Bokeh**	**Altair**
Abstraction level	low	high	high	low	high
Interactivity	limited	no	yes	yes	yes
Maturity	very high	high	high	high	high
Stars on GitHub	19.8k [55]	12.3k [56]	15.8k [57]	19.1k [58]	9.1k [59]

Table 2.3 Plotting frameworks comparison.

2.2.4 The Arguments for Plotly and Dash

For the purpose of this thesis, Plotly and Dash were chosen as the main tools for the development of Ash. The choice started with the selection of dashboarding framework. Dash was selected because it supports partial updating and is more mature than Shiny. Partial updating is especially important because dimensionality reduction algorithms described in the Section 1.4 are computationally expensive. That could likely be solved by caching the results, but size of matrices to be stored can be large.

In terms of plotting frameworks, Plotly was selected because of its integration with Dash, however, the choice was not as essential as the choice of a dashboarding framework. Many other plotting frameworks would suit the use case well, as should be apparent from the Section 2.2.3.

2.3 Architectural Challenges

The following section discusses selected challenges that were faced during the development of Ash. Internal representation of dendrograms in Ash is described and the way it relates to format of expected input data is explored. Once the input data format is addressed, the custom splitting algorithm and the way it led to custom re-implementation of Plotly's dendrogram function is described. The last section describes the deployment and three possible ways of using Ash.

2.3.1 Dendrogram Parsing and Representation

A custom parsing solution was developed to read and translate input data necessary for the dendrogram visualization. The file format was inspired by the output of R's `hclust` function, but internally Ash uses Scipy because it is embedded in the Plotly ecosystem. The parsing algorithm is capable of converting from R's `hclust` format to Scipy's dendrogram format and is published in a supplementary PyPI package. This format is also expected from user as they bring their own data to Ash.

Four files are expected:

- `heights.csv`

  Heights.csv file refers to the height attribute of dendrogram. The height attribute represents the heights of individual nodes (clusters) in the dendrogram in the form of a numeric vector. Each element of the vector is tied to a specific node. It has the same number of rows as merge.csv and denotes the value of the distance metric when two nodes (either observation or clusters) were merged together. Usually, the values are in ascending order, but it may not hold for every metric.

- `merge.csv`

  Merge.csv is a 2D matrix that describes the connections between clusters and observations, in other words what is merged together. 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 Section 1.3.2.

- `order.csv`

  Order.csv 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.

- `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.

Additionally, custom precomputed dimensionality reduction files can be provided. The files should contain relevant matrices, and they will serve for caching purposes if provided. Should the data not be found it is created by leveraging the sklearn library, and saved in the `reduced_dimensions` folder for later use.

2.3.2 Splitting Algorithm

Guided yet flexible way of assigning points to clusters is essential for enabling fast and precise cluster assignment. It gives an option to be very granular but does not tie the user's hand with restrictions. Responsibility for the final assignment is in the hands of the user.

To form clusters from a dendrogram, multiple methods exist. The three most common are:

- Thresholding the height
- Thresholding the amount of clusters

- Thresholding intra-cluster distance

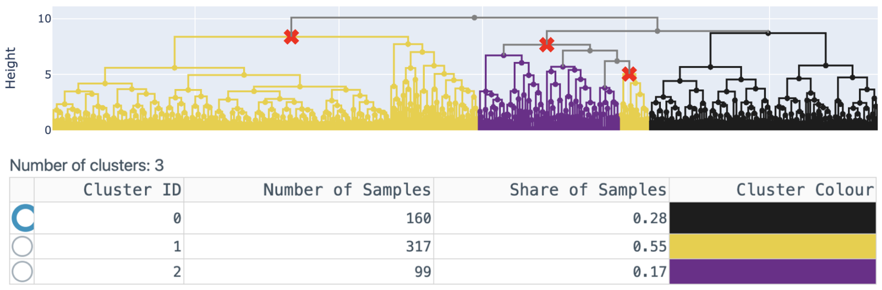
Thresholding the height means selecting target height, then every branch of the dendrogram under that height forms a separate cluster. Thresholding the amount of clusters selects a height threshold, such that the final number of clusters equals number of clusters desired. If the user wants individual distances between points within all clusters to not exceed chosen value, the intra-cluster distance thresholding is used. All three methods are standard with many implementations available.

The method that is implemented in this thesis aims to solve two issues with the previously mentioned methods:

- Apply thresholds only to a specific part of the dendrogram
- Allow co-assigning points to same cluster despite not being the closest in terms of the chosen metric.

Both of them are addressed by a simple algorithm. The user selects a node by either clicking on the dendrogram or by entering the node number and pressing the button for splitting. Then the dendrogram is traversed recursively starting from the selected node until the leaves are reached. Any leaf that is reached is assigned to the cluster specified by id entered by the user.

The algorithm allows a split at different heights on different branches, but having a meaningful split is the responsibility of the user. For instance, when the user selects a node that is a parent of previously selected node, the split will effectively be overwritten.



Number of clusters: 3

	Cluster ID	Number of Samples	Share of Samples	Cluster Colour
◉	0	160	0.28	⬛
○	1	317	0.55	🟨
○	2	99	0.17	🟪

Figure 2.1 A screenshot from Ash, depicting how to add a dendrogram node into an already existing cluster. Figure cropped for better visibility.

Thanks to the ability of *naming* clusters with unique IDs, Ash user can add a dendrogram node into *an already existing cluster*, provided the same ID is entered. This functionality is depicted in Figure 2.1, where the dendrogram is split in a way that cluster 1 (yellow) consists of two parts. The same figure also demonstrates that splitting can be done at different heights on different branches. Otherwise, the cluster 0 (black) would be split into parts as well. That is important because the chosen metric takes into account all features of the input data, which might not be the most suitable for particular cell types and because of the curse of dimensionality (see Section 1.3.2). This feature allows the user to form clusters disregarding the possibly inaccurate higher-level dendrogram structure created by the distance metric.

The algorithm is implemented in a recursive manner. Despite Python's lack of tail call optimization, the depth of the dendrogram is not expected to be large, so the recursion is not expected to be a problem.

2.3.3 Accessibility and Deployment

The user of a visualization tool should be able to use it without tedious installation process. In case of Ash, ease of access is addressed by the deployment as a standalone web application, but a docker image is also provided for situations where more computing power is needed.

One of planned accessibility features of Ash is that the user should be able to use the tool without programming knowledge. That is achieved by providing a simple dashboard-like interface that can be navigated by clicking on the dendrogram or buttons and selecting options from dropdown menus.

There are three options how to use Ash:

- Web application[1]
- Docker image
- Natively running the source code

The option with the lowest barrier to entry is through the website. Alternatively, the docker image can be downloaded. The image can be deployed on premise or to cloud via services like Kubernetes. The most hands-on approach is to interact with the code, which is hosted on github.com.

The decision to use Dash as a visualization framework enables web deployment without additional effort, as Dash creates the web server, runs it on the machine, and handles concurrent requests, user sessions and other challenging task. For the deployment, Heroku platform was selected. Heroku [60] is a serverless option that can scale in minutes without laborious migrations. The deployment process is fairly standardized, leaving just a few options for the user. That is an advantage for situation where the use-case is straightforward, but for less

[1]Accessible from https://ash-clustering-80f7adc27a4c.herokuapp.com/

straightforward use-cases, Heroku is not suitable. The ease of use is exchanged for the price. The price of the service is higher than virtual machine of equivalent performance. That leads to limitation of using the web app. Operations that are intensive on computing power, such as calculating t-SNE, will run much slower in the web application. For that reason the web application is suitable for experimentation with smaller datasets, exploring the application and the exploration of the application features. Once the dataset gets bigger, the other deployment variants, such as containerization of the native run, are more suitable.

The Docker image is a solution for situations where the user needs more computing power. The image is isolated from the environment of the host machine, which makes it transferable across different systems and machines, allowing frictionless deployment on a more powerful machine. The ideal situation in which the Docker image should be used is to deploy Ash for a limited group of users that knows their use case in advance. That could be, for instance, a lab.

If the user has programming skills, there are additional options to use Ash. The source code of Ash is available on github.com[2], where it can be downloaded, edited or users can run it as a Python script from their own computer. Lastly, a PyPI package can be installed, but the package only contains two utility functions - the splitting algorithm and conversion tool between R dendrograms and sklearn dendrograms.

2.4 User Interface

Ash's user interface aims to be simple and easy to navigate. On top of Ash, there is a header with three tabs pictured in Figure 2.2. The three tabs are *Instructions*, *Interactive Clustering*, *About*.

In Instructions, Ash and its components are explained. In About, information about Ash is provided. The main tab is Interactive Clustering. In this tab, all Ash's dashboards are available to the user.

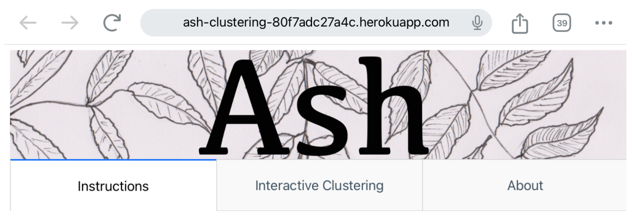


Figure 2.2 A screenshot from Ash, depicting the header and three tabs.

[2]Accessible at https://github.com/aemiliaurban/ash-interactive-dendrogram-tool

The following description refers to individual components found in Interactive Clustering.

The majority of options is available on the left side of Ash. The menus can be seen in Figure 2.4.

On top of the column, there is a button that allows the user to import their data. Clicking a button opens a pop-up window in which the user selects their files. It is necessary to select all files the user wants to import. The required files are described in the Section 2.3.1.

The menu that controls colors on all graphs in Ash is a simple drop-down menu that allows the user to turn colorblind palette on and off.

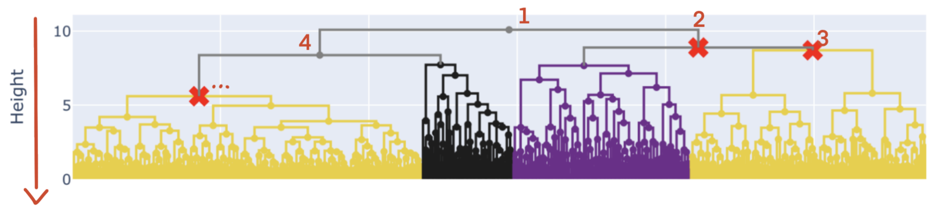
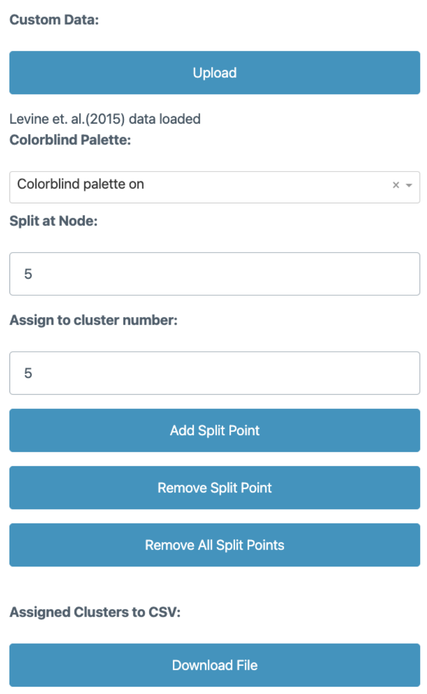


Figure 2.3 Demonstration of node numbering in Ash. The top node is number one and the number is increased with every other split as indicated by the orange arrow.

The two following input boxes are tied to the dendrogram which can be found on the right to the menu. Beneath “Split at Node”, there is an input text box which accepts integers. The input integer refers to the target node in the dendrogram, beneath which the split will be made. Nodes are numbered from top down as depicted in Figure 2.3. Furthermore, it is possible to assign the newly formed cluster to a user defined cluster by entering an integer in the second text input box. To add the split, the user needs to click on the “Add Split Point Button”. Splitting can be performed via the menu or by clicking on dendrogram nodes. Additionally, there is a button “Remove Split Point” that allows the user to remove a selected split point. It is also possible to clear current clusters by clicking on the “Remove All Split Points” button.

The last option in this part of the user interface is the “Download File” button which allows the user to save the newly clustered data to a *.csv* file. The file contains the original columns with an extra integer column which shows the final cluster.



Custom Data:

Upload

Levine et. al.(2015) data loaded

Colorblind Palette:

Colorblind palette on

Split at Node:

5

Assign to cluster number:

5

Add Split Point

Remove Split Point

Remove All Split Points

Assigned Clusters to CSV:

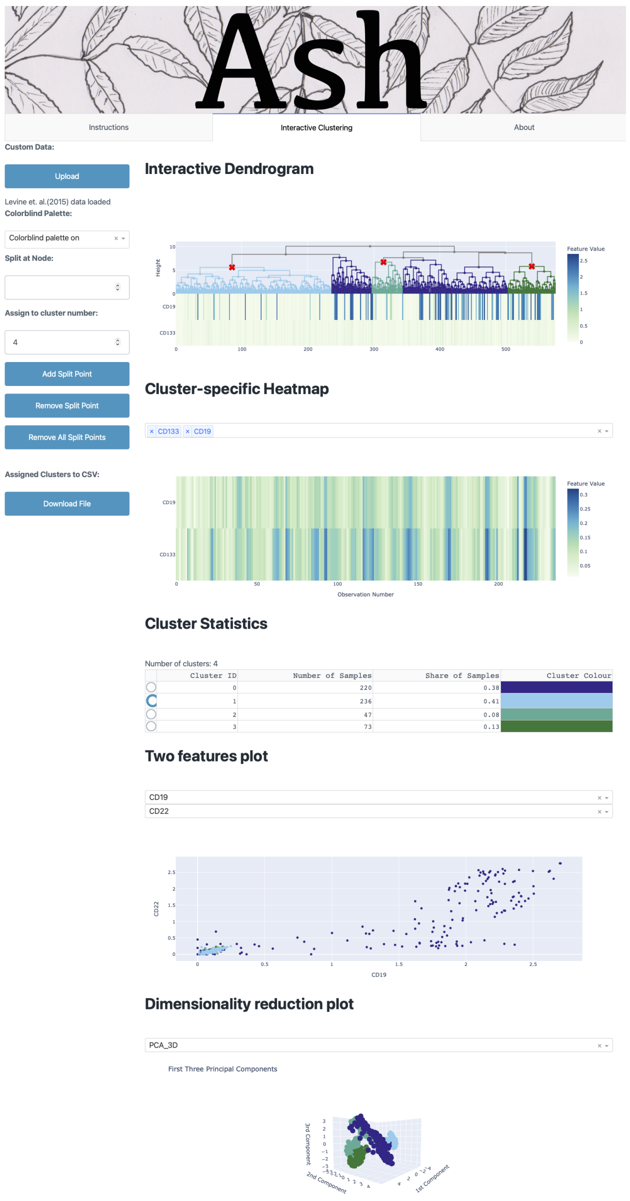
Download File

Figure 2.4 A screenshot from Ash, depicting available menus.

Figure 2.5 shows the available graphs in Ash. As described above, the menu ties to the available dendrogram representation. The data in the example were split into 4 clusters. Beneath the data points is a heatmap. The heatmap is connected to features selected in the drop-down menu of Cluster Specific Heatmap, and are presented in the same order as the end leaf nodes of the dendrogram. This allows the user to spot potential differences between the clusters. Only two features are selected in the example, however, it is possible to select more.

The next element shows the cluster specific heatmap. It is connected to the Cluster Statistics table below, which also serves as a radio button menu for selecting which specific cluster should appear on the cluster specific heatmap. Furthermore, the Cluster Statistics element allows the user to inspect the number and share of samples in each cluster.

The final two plots are the Two features plot and the Dimensiolanility reduction plot. The colors in these plots correspond to their assigned clusters in the dendrogram.



Custom Data:

Levine et. al.(2015) data loaded

Colorblind Palette: Colorblind palette on

Split at Node:

Assign to cluster number: 4

Assigned Clusters to CSV:

Interactive Dendrogram

Cluster-specific Heatmap

CD133, CD19

Cluster Statistics

Number of clusters: 4

	Cluster ID	Number of Samples	Share of Samples	Cluster Colour
○	0	220	0.38	■
●	1	236	0.41	■
○	2	47	0.08	■
○	3	73	0.13	■

Two features plot

CD19
CD22

Dimensionality reduction plot

PCA_3D

Figure 2.5 A screenshot from Ash which shows the available menus and plots.

Chapter 3

Case Study: Eosinophils in Samusik_all Dataset

In this chapter, Ash is used to analyze Samusik_all dataset [3]. The structure and characteristics of Samusik_all dataset are introduced in Section 3.1. Once the dataset is introduced, the heterogeneity among the cells labeled as eosinophils is highlighted. This heterogeneity is further explored using Ash with the aim of identifying two distinct clusters, likely representing distinct cell populations among the cells that should only be eosinophils, according to their manually gated labels. Marker levels of the two suspected populations are investigated and results are reported.

Based on the results of the analysis, alternative labels for the Samusik_all dataset are proposed, and they are compared to the original labels. By the end of this chapter, the reader should be familiar with the workflow of using Ash and what kind of insights can be gathered from it.

3.1 Dataset Characteristics

The dataset used for the analysis originates from Samusik et al. [3]. In their paper, Samusik et al. [3] are concerned with finding cell populations algorithmically with their novel X-shift algorithm. The authors demonstrate X-shift's performance on a dataset that was manually gated for the purposes of the paper. Three cytometry experts were tasked with identifying 24 cell populations in the dataset. The gating strategy of these experts was the same, but the gates were selected independently. Resulting labels that were assigned and published with the paper were obtained from the consensus among the three experts.

Two versions of the dataset were published. Samusik_all dataset contains all cells, while Samusik_01 contains data points from the sample 01 only. For the overview of both versions of the dataset, refer to the Table 3.1.

This thesis uses Samusik_all dataset for the analysis, but comparable results on Samusik_01 by Becht et al. [61] are briefly discussed in the following sections.

	Samusik_all dataset	Samusik_01 dataset
Number of cells	841 644	86 864
Number of dimensions	39	39
Number of manually gated population	24	24
Share of manually gated cells	61%	61%
Origin	10 *C57BL/6* mice	1 *C57BL/6* mouse

Table 3.1 Overview of the Samusik_all dataset.

3.2 Eosinophils in Samusik Datasets

Eosinophils are a type of white blood cells that is involved in the immune response to parasites and allergic reactions. This section investigates the heterogeneity among them in the Samusik datasets.

As of 2024, UMAP as well as t-SNE are commonly used to aid clustering for flow cytometry data. Examples can be found in Stolarek et al.[41] and Mair et al. [15]. Both methods are harder to interpret due to their non-linear nature, and because resulting projection depends on hyperparameters. Choice of hyperparameters influences whether the algorithms focus more on local structure or global structure of the data. In the example of the t-SNE cluster size as well as distances between clusters can be meaningless [62]. Despite the caution that must be taken when interpreting the results, the methods aim to maintain high dimensionality structure and map it to a lower dimension. For that reason, distinct cell populations are expected to be visually separated to a high degree.

In case of the Samusik_all dataset, eosinophils exhibit a concerning level of heterogeneity. See Figure 3.1 for the UMAP representation of the dataset with eosinophils highlighted. The yellow color represents manually gated eosinophils. Visual inspection suggests two distinct clusters among the cells labeled as eosinophils. Comparable heterogeneity was also reported by Becht et al. [61] on the Samusik_01 dataset. See Figure 3.2 for the UMAP and t-SNE with eosinophils highlighted.

In the Samusik datasets, eosinophils were gated using SiglecF and CD11b markers. The data points where both markers were high are labeled as eosinophils. See Figure 3.3 for the exact shape of the gate.

SiglecF and CD11b are commonly used markers for eosinophils, but other cells with high levels of these markers exist. One of the possible reasons for the

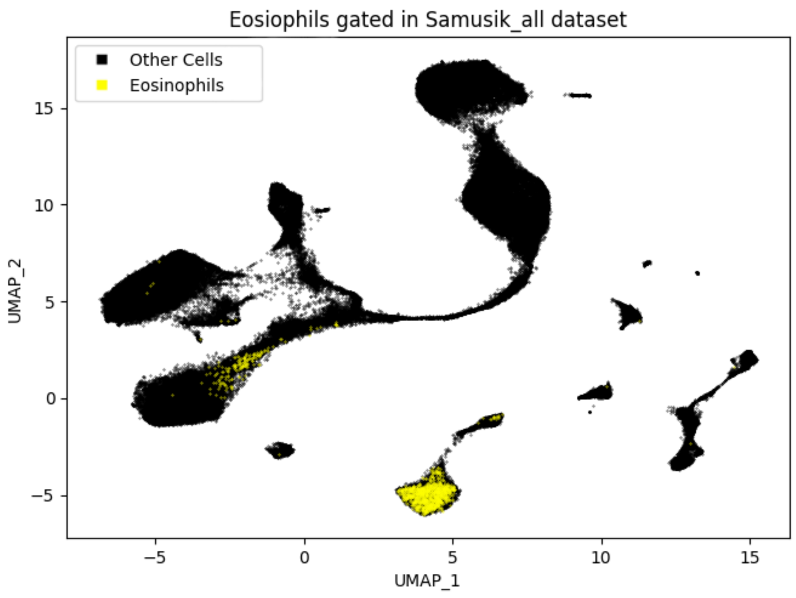


Figure 3.1 UMAP representation of the Samusik_all dataset with eosinophils highlighted.

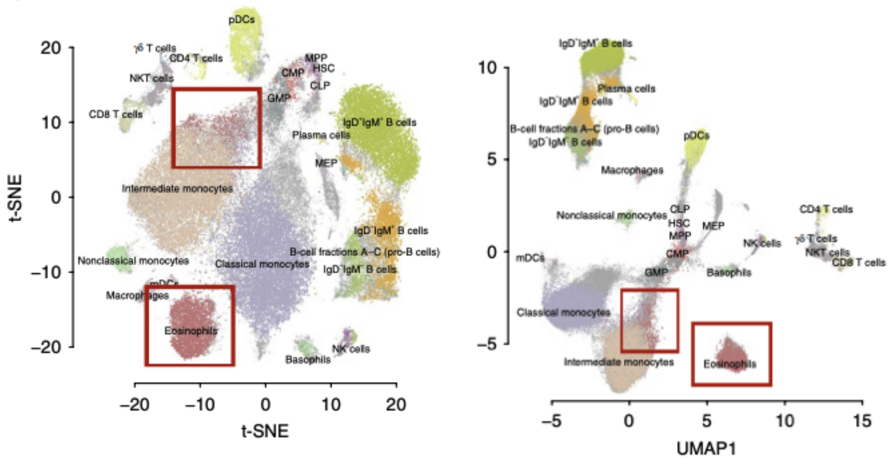


Figure 3.2 UMAP and t-SNE projections of the Samusik_01 dataset with eosinophils highlighted. Taken from Becht et al. [61]. Highlight by the author of this thesis.

heterogeneity in UMAP and t-SNE projections is that other cells with high SiglecF and CD11b were mistakenly considered to be eosinophils.

Neutrophils can be mistaken for eosinophils, as both cell types express CD15 and CD66 [63] but none of these markers are available in the Samusik datasets.

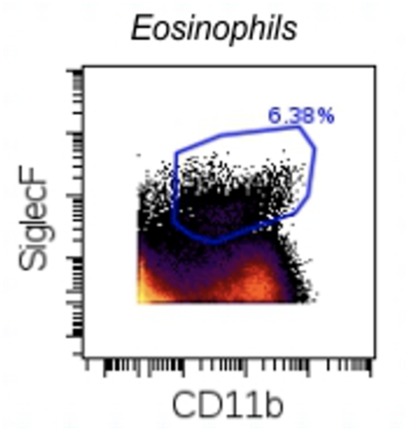


Figure 3.3 Manually gated eosinophils from Samusik_all dataset. CD11b and SiglecF were chosen as markers by the authors of the dataset. Taken from supplementary materials [3].

Bolden et al. [64] suggest that CD11b+ SiglecF+ IL5Rα- cells develop into non-eosinophil granulocytes in C57BL/6 mice. But as of 2024, the research on other CD11 and SiglecF non-eosinophils in mouse bone marrow is limited. For the overview of markers that are used to identify eosinophils see the Table A.1.

Instead of identifying eosinophils by markers they express, lack of expression can be used as supportive evidence as well. According to this article on Life Sciences [63] and Thurau et al. [65], eosinophils lack CD16 which is a marker found on the surface of NK cells and monocytes [66]. See the Figure 3.4 from Thurau et al. [65] showing the lack of CD16 expression. Samusik datasets do not contain CD16 marker, but they contain CD16/32.

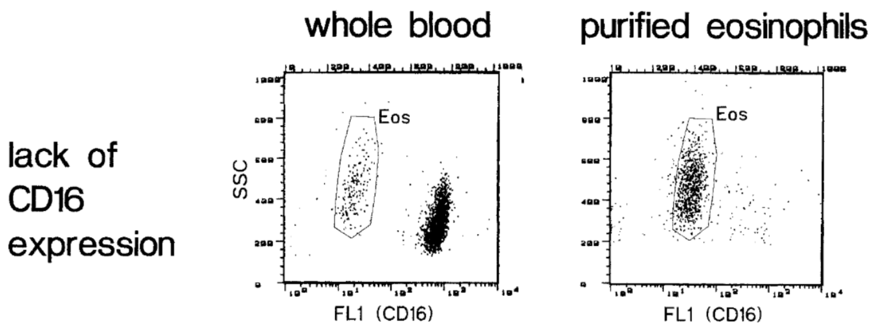


Figure 3.4 Lack of CD16 expression on eosinophils. Taken from Thurau et al. [65].

Next section describes how Ash was used to identify two distinct clusters among the cells labeled as eosinophils in Samusik_all dataset.

3.3 Clusters Among Eosinophils

This section investigates the heterogeneity among the cells labeled as eosinophils in the Samusik_all dataset.

3.3.1 Methodology

The analysis was applied only to eosinophils from the Samusik_all dataset. Before the analysis, the dataset was split into 11 parts to ensure the plots remain clear and legible. Each part but the last consists of 5 000 cells and R `hclust` was used to generate dendrogram data in the required format. For each part of the dataset the following procedure was applied:

Load data to Ash

For performance reasons, the version of Ash that was used ran on a local machine with Apple M1 chip and 16 GB of RAM. Output of R `hclust` was exported to a .csv file and was loaded to Ash together with the dataset.

Cluster data

Once the data was loaded, UMAP plot was used to guide the clustering. Target number of clusters was set to 2, as eosinophils were observed to form two distinct clusters on the Figure 3.1. Then node or nodes that achieved the best separation on the UMAP plot were selected to split the data. Nodes, where dendrograms were split, Ash configuration and resulting clusters are all available on GitHub [1]. See Figure 3.5 for an example of the resulting split.

Inspect and identify discriminating markers

The aim of this step was to interact with the scatter plot and heatmap with the goal of identifying markers that are different between the two clusters. SiglecF and CD11b were expected to have high values, as their high values was the criteria for data points to be labeled as eosinophils. Special attention was paid to CD16/32, as it is expected to be absent in eosinophils [65] and it was not part of the gating process. For the example of different levels among the two clusters refer to the Figure 3.6.

[1]Accessible from https://github.com/aemiliaurban/ash-interactive-dendrogram-tool

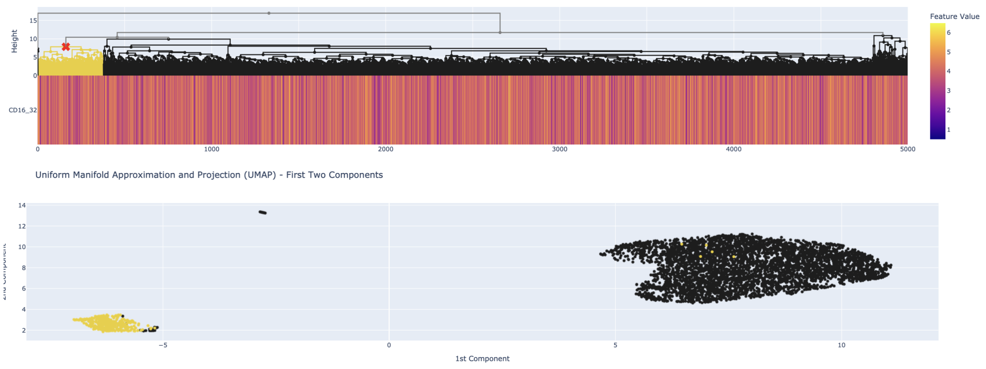


Figure 3.5 UMAP guided clustering of the 9th part of the Samusik_all dataset. The Figure was generated in Ash and cropped for the improved visibility.

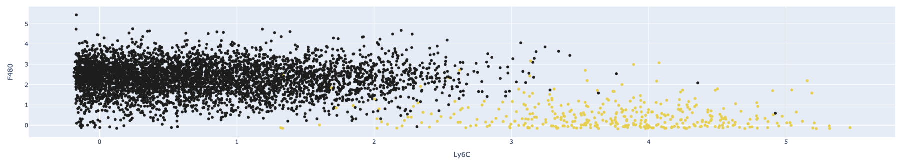


Figure 3.6 Scatter plot of Ly6C (on horizontal axis) and F480 (on vertical axis) markers from part 9 of the Samusik_all dataset.

Label clusters based on CD16/32

This step starts with downloading the data from Ash through the export button. The data is further processed so that clusters are not labeled by their index, but by the mean CD16/32 level (low or high). The goal of such labeling is to be able to combine all parts of the dataset back together to reconstruct the UMAP plot (see Figure 3.1) with the clusters labeled by their CD16/32 levels.

Reconstruct the UMAP with labeled clusters

In case of the data mislabeling, it is expected to observe clusters tightly packed together with minimal mixing between them. The cluster with low CD16/32 is expected to be correctly labeled eosinophils, while the cluster with high CD16/32 is expected to be mislabeled cells.

3.3.2 Results

Major attention was paid to CD16/32 as it is expected to be absent on eosinophils [65], but other markers were noticeably different for both clusters among all parts of the dataset. In general, all data parts exhibited very similar patterns with only limited variation across different parts. The markers that were consistently different between the two clusters were CD16/32, F480, Ly6C. For the reference on how noticeable the difference between markers level was, please refer to the Figure 3.6. The Figure demonstrates the scatter plot of F480 and Ly6C on the ninth part of the data. In this particular case it is clearly visible that t-SNE guided clusters separate the population almost perfectly.

Detailed differences in means of CD16/32, F480, Ly6C between clusters can be inspected in the Tables B.1, B.2, B.3 respectively. Note that the Table B.1 should be interpreted with caution. Cluster labeled as cluster with low mean levels of CD16/32 has low mean levels of CD16/32 by design.

Summarized levels of the markers among combination of all data parts can be inspected on Figure 3.7. It is visible that group with lower CD16/32 levels has also lower F480 but higher Lyc6 marker levels.

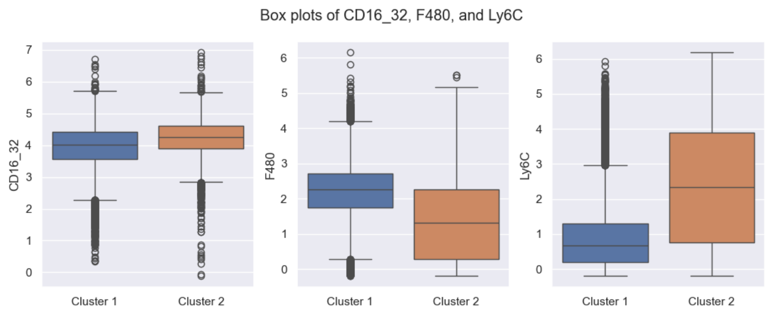


Figure 3.7 Boxplot of CD16/32, F480, and Ly6C markers for the combined clusters with low and high CD16/32. Cluster 1 respresent the cluster with low mean CD16/32, cluster 2 represents the cluster with high mean CD16/32.

Ly6c is an antigen commonly used to differentiate between classical and non-classical monocytes/macrophages [67] and F480 is a marker for macrophages [68]. In the context of previously observed heterogeneity that could mean that the cluster with lower levels of CD16/32 could be true eosinophils as we expect low values of CD16/32 [65]. The other cluster's higher value of Ly6C would suggest it might either be a population of monocytes or a population of macrophages. Argument against macrophages would be the relatively lower level F480 among the cells in the cluster. These observations suggest that the heterogeneity among eosinophils could be explained by monocytes being mistaken for eosinophils.

When all data parts are combined back with the original Samusik_all dataset, a UMAP plot can be formed. Compared to the Figure 3.1, UMAP on combined dataset allows highlighting both clusters that were formed in Ash. If the heterogeneity from the Figure 3.1 is caused by monocytes, the suspicious points (ones that visually does not belong to the cluster of eosinphils) are expected to have higher CD16/32, lower F480 and higher Ly6C. The aforementioned UMAP is depicted in the Figure 3.8. Suspected monocytes form their separate cluster, that is visually distinct from the rest of the eosinophils. That further support the hypothesis of mislabeling.

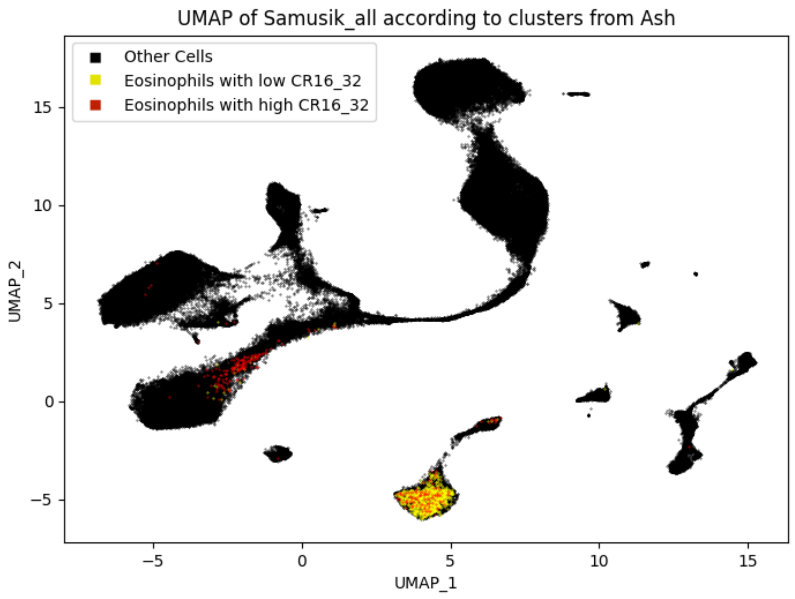


Figure 3.8 UMAP representation of the Samusik_all dataset clusters formed in Ash.

The analysis in this chapter demonstrates, how a visualization and clustering tool can help to discover the irregularities in the data. As this analysis only compared relative levels of selected markers, further research is encouraged, and it can be built upon the foundation of this chapter. Along with this thesis an alternative cellular population labels for Samusik_all are published, where eosinophils are split to the aforementioned clusters based UMAP-guided clustering procedure described in this chapter and performed in Ash. The labels are available from the Author's Github.

Conclusion

This thesis describes Ash, a visualization and clustering tool for flow cytometry data analysis.

The first chapter introduces flow cytometry, covering its fundamental principles, data outputs, and currently available analysis techniques, including manual analysis techniques and automated clustering methods. The last part of the chapter is dedicated to data visualization in flow cytometry.

The second chapter dives into currently available clustering tools in flow cytometry. It focuses on Ash, the proprietary visualization tool for flow cytometry data analysis developed in this thesis. It covers the selected technologies, architecture and architectural challenges, and user interface. Ash is designed to allow interactive and intuitive visualization and clustering of preprocessed data. It offers a variety of plots including interactive dendrogram, which can be used to split data into clusters interactively by the user. The dendrogram is connected to a heatmap, which can help the user see patterns in the data. Furthermore, it is possible to plot two features of user's choice on a scatter plot, and see the data in plots based on dimensionality reduction algorithms PCA, t-SNE, and UMAP. The algorithms are available in both 2D and 3D.

To validate Ash's effectiveness, the benchmark Samusik_all dataset was analysed. The data is tested under the hypothesis that using data visualization techniques can reveal underlying patterns and help with data analysis. The analysis dicovered an incorrect labeling of one of the cell populations. The cells that were labeled as eosinophils by manual gating are a heterogeneious group. The samples in this group express different CD16/32 levels, markers that can be found in neutrophils but not in eosinophils. However, the levels of markers F480 and Ly6C indicate that the cells might be monocytes. The separation of these two cell populations is visible on the UMAP plot, in which the two populations are located in different areas of the plot.

The outcomes of this thesis highlight the potential of powerful data visualization tools such as Ash as key parts of flow cytometry data analysis. By providing user-friendly and customizable visualization options, researchers can interact with data in more efficient ways, leading to more accurate and reliable results.

The main contribution of this thesis lies in the development of a user-friendly and open-source tool that can be used by flow cytometry researchers to improve their data analysis for free. The free availability of Ash can hopefully encourage the wider adoption of data visualization techniques among those with limited programming experience.

Bibliography

[1] Adam Šmelko et al. "GPU-Accelerated Mahalanobis-Average Hierarchical Clustering Analysis". In: *Euro-Par 2021: Parallel Processing: 27th International Conference on Parallel and Distributed Computing, Lisbon, Portugal, September 1–3, 2021, Proceedings 27.* Springer. 2021, pp. 580–595.

[2] Tomáš Sieger et al. "Interactive dendrograms: the R packages idendro and idendr0". In: *Journal of Statistical Software* 76 (2017), pp. 1–22.

[3] Nikolay Samusik et al. "Automated mapping of phenotype space with single-cell data". In: *Nature methods* 13.6 (2016), pp. 493–496.

[4] Katherine M McKinnon. "Flow cytometry: an overview". In: *Current protocols in immunology* 120.1 (2018), pp. 5–1.

[5] Pearlson P Austin Suthanthiraraj and Steven W Graves. "Fluidics". In: *Current Protocols in Cytometry* 65.1 (2013), pp. 1–2.

[6] Aysun Adan et al. "Flow cytometry: basic principles and applications". In: *Critical reviews in biotechnology* 37.2 (2017), pp. 163–176.

[7] Stephen P Perfetto et al. "Quality assurance for polychromatic flow cytometry". In: *Nature protocols* 1.3 (2006), pp. 1522–1530.

[8] Carleton C Stewart. "Multiparameter flow cytometry". In: *Journal of Immunoassay* 21.2-3 (2000), pp. 255–272.

[9] AL Givan. "Critical Aspects of Staining for Flow Cytometry". In: *Living Color: Protocols in Flow Cytometry and Cell Sorting* (2000), pp. 142–164.

[10] Laura Ferrer-Font et al. "Panel design and optimization for high-dimensional immunophenotyping assays using spectral flow cytometry". In: *Current protocols in cytometry* 92.1 (2020), e70.

[11] Josef Spidlen et al. "Data file standard for flow cytometry, version FCS 3.2". In: *Cytometry Part A* 99.1 (2021), pp. 100–102.

[12] John Drakos et al. "A perspective for biomedical data integration: Design of databases for flow cytometry". In: *BMC bioinformatics* 9 (Feb. 2008), p. 99. DOI: 10.1186/1471-2105-9-99.

[13] Holden T Maecker et al. “Standardization of cytokine flow cytometry assays”. In: *BMC immunology* 6 (2005), pp. 1–18.

[14] Holden T Maecker, J Philip McCoy, and Robert Nussenblatt. “Standardizing immunophenotyping for the human immunology project”. In: *Nature Reviews Immunology* 12.3 (2012), pp. 191–200.

[15] Florian Mair et al. “The end of gating? An introduction to automated analysis of high dimensional cytometry data”. In: *European journal of immunology* 46.1 (2016), pp. 34–43.

[16] Greg Finak et al. “Merging mixture components for cell population identification in flow cytometry”. In: *Advances in bioinformatics* 2009 (2009).

[17] Florian Kowarsch et al. “Towards Self-explainable Transformers for Cell Classification in Flow Cytometry Data”. In: *Interpretability of Machine Intelligence in Medical Image Computing: 5th International Workshop, iMIMIC 2022, Held in Conjunction with MICCAI 2022, Singapore, Singapore, September 22, 2022, Proceedings*. Springer. 2022, pp. 22–32.

[18] *Clustering Algorithms*. `https://developers.google.com/machine-learning/clustering/clustering-algorithms`. [Accessed 21-07-2024].

[19] Shihyen Chen, Bin Ma, and Kaizhong Zhang. “On the similarity metric and the distance metric”. In: *Theoretical Computer Science* 410.24-25 (2009), pp. 2365–2376.

[20] Vijay Kumar, Jitender Kumar Chhabra, and Dinesh Kumar. “Performance evaluation of distance metrics in the clustering algorithms”. In: *INFOCOMP Journal of Computer Science* 13.1 (2014), pp. 38–52.

[21] Archana Singh, Avantika Yadav, and Ajay Rana. “K-means with Three different Distance Metrics”. In: *International Journal of Computer Applications* 67.10 (2013).

[22] Karel Fišer et al. “Detection and monitoring of normal and leukemic cell populations with hierarchical clustering of flow cytometry data”. In: *Cytometry Part A* 81.1 (2012), pp. 25–34.

[23] Andrew Cron et al. “Hierarchical modeling for rare event detection and cell subset alignment across flow cytometry samples”. In: *PLoS computational biology* 9.7 (2013), e1003130.

[24] Tom Ronan, Zhijie Qi, and Kristen M Naegle. “Avoiding common pitfalls when clustering biological data”. In: *Science signaling* 9.432 (2016), re6–re6.

[25] Hadi Shiravi, Ali Shiravi, and Ali A Ghorbani. “A survey of visualization systems for network security”. In: *IEEE Transactions on visualization and computer graphics* 18.8 (2011), pp. 1313–1329.

[26] Nemshan Alharthi, Adnan Gutub, et al. “Data visualization to explore improving decision-making within Hajj services”. In: *Scientific Modelling and Research* 2.1 (2017), pp. 9–18.

[27] Kishore Singh and Peter Best. “Anti-money laundering: Using data visualization to identify suspicious activity”. In: *International Journal of Accounting Information Systems* 34 (2019), p. 100418.

[28] Mingchen Feng et al. “Big data analytics and mining for effective visualization and trends forecasting of crime data”. In: *IEEE Access* 7 (2019), pp. 106111–106123.

[29] Seungeun Park et al. “Impact of data visualization on decision-making and its implications for public health practice: a systematic literature review”. In: *Informatics for Health and Social Care* 47.2 (2022), pp. 175–193.

[30] Tricia Aung, Debora Niyeha, and Rebecca Heidkamp. “Leveraging data visualization to improve the use of data for global health decision-making”. In: *Journal of global health* 9.2 (2019).

[31] Graham Dove et al. “Using data visualization in creativity workshops: a new tool in the designer’s kit”. In: *Proceedings of the 9th ACM Conference on Creativity & Cognition.* 2013, pp. 304–307.

[32] Qi Li. “Data visualization as creative art practice”. In: *Visual Communication* 17.3 (2018), pp. 299–312.

[33] Tracey L Weissgerber et al. “From static to interactive: transforming data visualization to improve transparency”. In: *PLoS biology* 14.6 (2016), e1002484.

[34] Jacob H Levine et al. “Data-driven phenotypic dissection of AML reveals progenitor-like cells that correlate with prognosis”. In: *Cell* 162.1 (2015), pp. 184–197.

[35] Tadeusz CaliŃski. “Dendrogram”. In: *Wiley StatsRef: Statistics Reference Online* (2014).

[36] Hyekyoung Lee et al. “Persistent brain network homology from the perspective of dendrogram”. In: *IEEE transactions on medical imaging* 31.12 (2012), pp. 2267–2277.

[37] EW Rosolowsky et al. “Structural analysis of molecular clouds: dendrograms”. In: *The Astrophysical Journal* 679.2 (2008), p. 1338.

[38] Richard R Jahan-Tigh et al. “Flow cytometry”. In: *The Journal of investigative dermatology* 132.10 (2012), e1.

[39] Enrico Lugli et al. "Subject classification obtained by cluster analysis and principal component analysis applied to flow cytometric data". In: *Cytometry Part A: The Journal of the International Society for Analytical Cytology* 71.5 (2007), pp. 334–344.

[40] Shadi Toghi Eshghi et al. "Quantitative comparison of conventional and t-SNE-guided gating analyses". In: *Frontiers in immunology* 10 (2019), p. 1194.

[41] Ireneusz Stolarek et al. "Dimensionality reduction by UMAP for visualizing and aiding in classification of imaging flow cytometry data". In: *Iscience* 25.10 (2022).

[42] *flowjo*. `https://www.flowjo.com/`. Accessed: 2024-07-05.

[43] *flowlogic*. `https://flowlogic.software/`. Accessed: 2024-07-05.

[44] *Cytobank*. `https://www.beckman.com/flow-cytometry/software/cytobank-premium/`. Accessed: 2024-07-05.

[45] *AUTOKLUS*. `http://www.cyto.purdue.edu/flowcyt/software/Catalog.html`. Accessed: 2024-07-05.

[46] *idendro*. `https://github.com/tsieger/idendro`. Accessed: 2024-07-05.

[47] Sehi L'Yi et al. "XCluSim: a visual analytics tool for interactively comparing multiple clustering results of bioinformatics data". In: *BMC bioinformatics* 16 (2015), pp. 1–15.

[48] Subhajit Das et al. "Geono-cluster: Interactive visual cluster analysis for biologists". In: *IEEE Transactions on Visualization and Computer Graphics* 27.12 (2020), pp. 4401–4412.

[49] X FlowJo. "v10. 0.7 r2 FlowJo". In: *LLC https://www. flowjo. com* (2020).

[50] Tom C Bakker Schut, Bart G De Grooth, and Jan Greve. "Cluster analysis of flow cytometric list mode data on a personal computer". In: *Cytometry: The Journal of the International Society for Analytical Cytology* 14.6 (1993), pp. 649–659.

[51] *Tiobe Statistic*. `https://www.tiobe.com/tiobe-index/`. Accessed: 2024-07-05.

[52] *Dash on Github*. `https://github.com/plotly/dash`. Accessed: 2024-07-05.

[53] *Python Shiny on Github*. `https://github.com/posit-dev/py-shiny`. Accessed: 2024-07-05.

[54] *Streamlit on Github*. `https://github.com/streamlit/streamlit`. Accessed: 2024-07-05.

[55] *Matplotlib on Github*. https://github.com/matplotlib/matplotlib. Accessed: 2024-07-05.

[56] *Seaborn on Github*. https://github.com/mwaskom/seaborn. Accessed: 2024-07-05.

[57] *Plotly on Github*. https://github.com/plotly/plotly.py. Accessed: 2024-07-05.

[58] *Boteh on Github*. https://github.com/bokeh/bokeh. Accessed: 2024-07-05.

[59] *Altair on Github*. https://github.com/vega/altair. Accessed: 2024-07-05.

[60] *Cloud Application Platform | Heroku — heroku.com*. https://www.heroku.com/. [Accessed 01-08-2024].

[61] Etienne Becht et al. "Dimensionality reduction for visualizing single-cell data using UMAP". In: *Nature biotechnology* 37.1 (2019), pp. 38–44.

[62] Martin Wattenberg, Fernanda Viégas, and Ian Johnson. "How to Use t-SNE Effectively". In: *Distill* (2016). DOI: 10.23915/distill.00002. URL: http://distill.pub/2016/misread-tsne.

[63] *Eosinophils*. https://www.beckman.com/resources/cell-types/blood-cells/leukocytes/eosinophils. Accessed: 2024-06-30.

[64] Jessica E Bolden et al. "Identification of a Siglec-F+ granulocyte-macrophage progenitor". In: *Journal of leukocyte biology* 104.1 (2018), pp. 123–133.

[65] Anja Maria Thurau et al. "Identification of eosinophils by flow cytometry". In: *Cytometry: The Journal of the International Society for Analytical Cytology* 23.2 (1996), pp. 150–158.

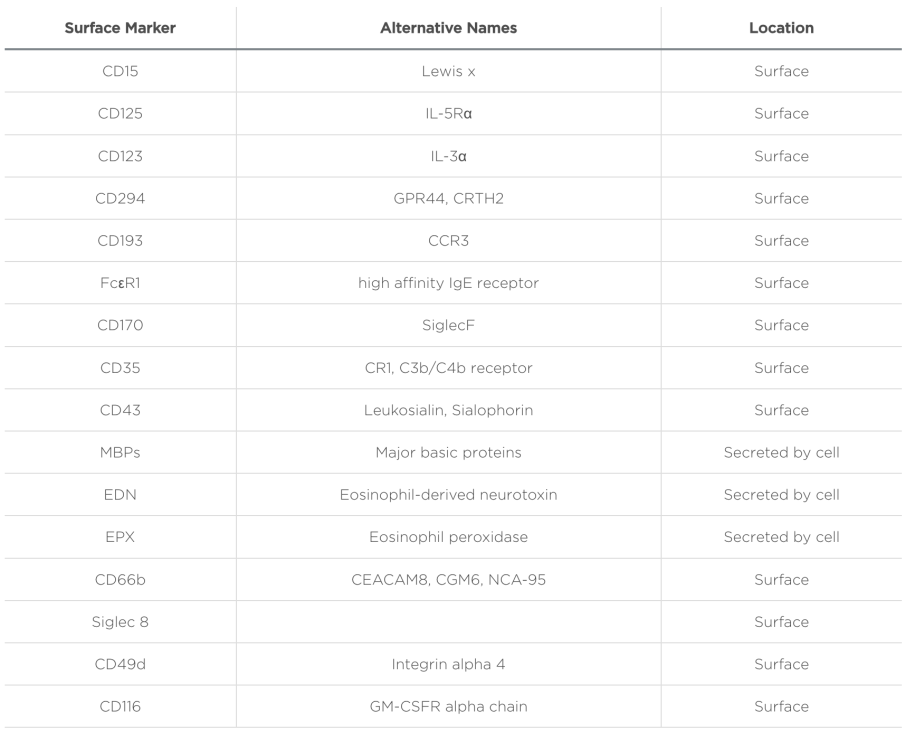
[66] Wei Hseun Yeap et al. "CD16 is indispensable for antibody-dependent cellular cytotoxicity by human monocytes". In: *Scientific reports* 6.1 (2016), p. 34310.

[67] Marina Martínez-Carmona et al. "Ly6c as a new marker of mouse blood vessels: Qualitative and quantitative analyses on intact and ischemic retinas". In: *International Journal of Molecular Sciences* 23.1 (2021), p. 19.

[68] Alexandra dos Anjos Cassado. "F4/80 as a major macrophage marker: the case of the peritoneum and spleen". In: *Macrophages: Origin, functions and biointervention* (2017), pp. 161–179.

Appendix A

Markers used to identify Eosinophils



Surface Marker	Alternative Names	Location
CD15	Lewis x	Surface
CD125	IL-5Rα	Surface
CD123	IL-3α	Surface
CD294	GPR44, CRTH2	Surface
CD193	CCR3	Surface
FcεR1	high affinity IgE receptor	Surface
CD170	SiglecF	Surface
CD35	CR1, C3b/C4b receptor	Surface
CD43	Leukosialin, Sialophorin	Surface
MBPs	Major basic proteins	Secreted by cell
EDN	Eosinophil-derived neurotoxin	Secreted by cell
EPX	Eosinophil peroxidase	Secreted by cell
CD66b	CEACAM8, CGM6, NCA-95	Surface
Siglec 8		Surface
CD49d	Integrin alpha 4	Surface
CD116	GM-CSFR alpha chain	Surface

Figure A.1 Markers used to identify eosinophils. Taken from Life Sciences [63].

Appendix B

Difference in Marker levels among UMAP-guided Clusters of Eosinophils

Mean CB16/32 levels		
Data Part	**Cluster with low CB16/32**	**Cluster with high CB16/32**
Part 1	3.868	4.126
Part 2	3.82	4.237
Part 3	3.906	4.174
Part 4	4.041	4.158
Part 5	4.133	4.413
Part 6	4.046	4.31
Part 7	4.104	4.276
Part 8	3.896	4.224
Part 9	3.961	4.121
Part 10	4.059	4.386
Part 11	3.785	4.284

Table B.1 Mean levels of CD16/32 for each data part

Mean F480 levels		
Data Part	**Cluster with low CB16/32**	**Cluster with high CB16/32**
Part 1	2.211	0.682
Part 2	2.324	0.71
Part 3	2.239	0.683
Part 4	1.583	2.181
Part 5	2.184	0.962
Part 6	1.888	0.621
Part 7	2.199	0.532
Part 8	2.331	0.981
Part 9	2.315	0.472
Part 10	2.214	1.036
Part 11	2.398	1.448

Table B.2 Mean levels of F480 for each data part

Mean Ly6C levels		
Data Part	**Cluster with low CB16/32**	**Cluster with high CB16/32**
Part 1	0.905	4.057
Part 2	0.931	3.14
Part 3	0.932	3.245
Part 4	2.197	0.877
Part 5	0.749	3.048
Part 6	0.641	3.206
Part 7	0.833	3.846
Part 8	0.749	3.345
Part 9	0.756	3.731
Part 10	0.788	3.025
Part 11	0.644	2.606

Table B.3 Mean levels of Ly6C for each data part

Appendix C

Local installation guide

Ash is freely available on GitHub, and can be downloaded from its cloud-based repository hosting service at

```
https://github.com/aemiliaurban/ash-interactive-dendrogram-tool
```

The applications comes with demo data included. The source of the data is the Levine dataset [34]. There are multiple ways the project can be downloaded to the end user's computer.

It is possible to use the git clone command for users who have pre-established HTTPS or SSH connections to GitHub, however, setting up these connections is out of the scope of this thesis.

For users without pre-established GitHub connections, it is possible to download the application as .zip by clicking on the "<> Code" button and choosing the option "Download ZIP", as represented in figure C.1 by green square. Once the .zip file is downloaded, unzip the files to location of choice, and the installation is complete.

Ash supports python3.10. Make sure you have it installed on your machine. It can be installed from terminal with homebrew:

```
brew install python@3.10
```

OPTIONAL: creating a virtual environment (highly recommended):

```
python3.10 -m venv venv
```

Activate the virtual environment (operation system dependent):

```
source venv/bin/activate
```

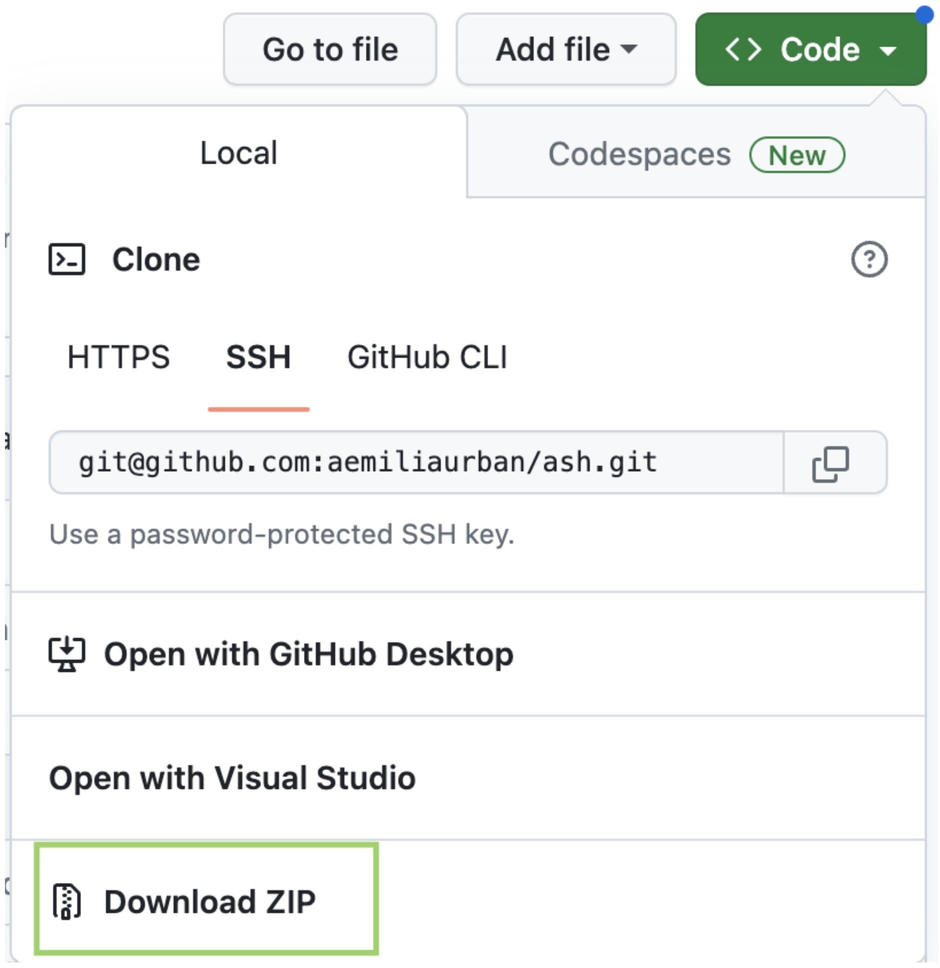
Install requirements:

```
pip install -r requirements.txt
```

Run the app: Before running the app, make sure you are in the ash directory (relative paths).

```
python app.py
```

The virtual environment has to be activated via a command. Activation of the virtual environment is system and shell dependent, as shown in table C.1.



```
git@github.com:aemiliaurban/ash.git
```

Figure C.1 Download menu on GitHub.

Platform	**Shell**	**Command to activate virtual environment**
POSIX	bash/zsh	source <venv>/bin/activate
POSIX	fish	source <venv>/bin/activate.fish
POSIX	csh/tcsh	source <venv>/bin/activate.csh
POSIX	PowerShell	<venv>/bin/Activate.ps1
Windows	cmd.exe	<venv>\Scripts\activate.bat
Windows	PowerShell	<venv>\Scripts\Activate.ps1

Table C.1 Virtual environment activation commands for individual systems and their shells.

Appendix D

Building Docker Image

It is also possible to run Ash from the included Dockerfile. Commands ought to be run from the root of the repo. Alternatively, you can adjust the paths.

Build the image:

```
docker build .  -t ash -no-cache
```

Run the container:

```
docker run -p 8050:8050 ash
```