A Comparison of Two Single-Cell RNA-Sequencing Analysis Pipelines using Snakemake

Pascal Meinhard

A thesis presented for the degree of
Bachelor of Science

Algorithmic Bioinformatics
Heinrich Heine University Düsseldorf
Germany

9th May, 2021

Acknowledgements

I am grateful to Prof. Dr. Gunnar Klau for giving me the opportunity to write this thesis and the extensive feedback on my writing. I thank Dr. Tobias Lautwein for being the second assessor and providing the data for testing. Special thanks to My Ky Huynh for providing one of the single-cell RNA sequencing analysis pipelines to use in this thesis and for answering my many questions.

Abstract

Single-cell RNA-sequencing provides us with access to the gene expression profiles of individual cells. However, due to technical confounders and unreliable cells, this data needs to be processed before it can be evaluated by the many available options for analysis. In this thesis, we compare two single-cell RNA-sequencing analysis workflows implemented as pipelines using *Snakemake*. Their main analysis components are the R packages *Seurat* and *Pagoda2*, respectively. After running the two pipelines on various data sets and evaluating the results, we concluded that they are similar in structure, but their analysis tools make them suitable for different research questions, that is, comparative or pathway analysis.

Contents

List of Tables

List of Figures

List of Abbreviations

scRNA-seq	Single-cell RNA-sequencing
FACS	Fluorescence-activated cell-sorting
PCR	Polymerase chain reaction
bp	Base pair
UMI	Unique molecular identifier
RNA	Ribonucleic acid
cDNA	Complementary deoxyribonucleic acid
QC	Quality control
PCA	Principal component analyis
PC	Principal component
t-SNE	t-distributed stochastic neighbor embedding
UMAP	Uniform manifold approximation and projection
KNN graph	k-nearest neighbor graph
DAG	Directed acyclic graph
GO	Gene Ontology
CSV	Comma-separated values
pANN	Proportion of artificial k-nearest neighbors
SNN graph	Shared nearest neighbor graph
PBMCs	Peripheral blood mononuclear cells
RI	Rand Index

1 Introduction

Single-Cell RNA-Sequencing (scRNA-seq) analysis is used to gain insight into the transcriptomes of individual cells and is thereby providing the necessary data to study the cellular heterogeneity of biological tissue samples. For instance, by clustering the cells, we get to uncover cellular subpopulations and infer marker genes, that would be hidden in bulk RNA-Seq analysis. However, before the data can be analyzed, it must be preprocessed to mitigate sampling effects from sequencing or to remove unreliable cells from it. Therefore, due to the many steps involved in the analysis of scRNA-seq data, it makes sense to build automated pipelines for it.

Additionally, with increasing interest in the field of scRNA-seq, the number of dedicated analysis tools is also growing[1]. While these tools serve a similar purpose, they can provide different analysis applications or perform better or worse than others. This makes the comparison of those tools worthwhile and also motivates the topic of this bachelor thesis: The comparison of two scRNA-seq analysis pipelines implemented with the workflow management system Snakemake (Mölder et al. 2021).

For this thesis, we present an scRNA-seq analysis pipeline, whose main component is the R package Pagoda2 (Barkas et al. 2021), and compare it to another Snakemake analysis pipeline that relies on the R package Seurat (Hao et al. 2020). To do so, we evaluate their differences and similarities by elaborating on their structure and compare their results in preprocessing and cluster analysis.

We concluded that the pipelines are similar in their approach, but their options in analysis make them applicable for different research questions and types of data. Seurat is more suitable for comparative analysis of scRNA-seq data with different conditions, such as healthy and diseased tissue, while Pagoda2 provides insights into biological processes in cells with pathway overdispersion analysis.

[1] *Approximately 921 tools, as of 16th April, 2021 (Zappia, Phipson, and Oshlack 2018)*

2 Background Information

In the following section, we introduce the necessary background knowledge for this thesis.

2.1 Single-Cell RNA-Sequencing

Single-cell RNA-sequencing (scRNA-seq), first introduced in 2009 (Tang et al. 2009), allows us to obtain the gene expression profiles of individual cells, providing a more detailed look at cellular heterogeneity than bulk RNA-sequencing.

To achieve this, the first step is to isolate the cells. There are several ways to do this, such as plate-based or droplet-based techniques. Plate-based techniques isolate cells by placing them into separate wells on a plate, often using fluorescence-activated cell sorting (FACS). However, this method is time-consuming and not efficient, as every plate has to be processed individually (Hwang, Lee, and Bang 2018). Since we are working with data in this thesis that was sequenced with the droplet-based Chromium plattform by *10x Genomics* (Zheng et al. 2017), we now explain its functionality in more detail.

Droplet-based approaches, such as Chromium 10x, offer high-throughput sequencing and thus allow us to process tens of thousands of cells at once. This is done by isolating cells with oil-encapsulated droplets in emulsion, as shown in Figure 1A. Every droplet contains a gel bead equipped with many oligonucleotides that are used to capture the transcripts of each cell. These oligomers consist of multiple components (Figure 1B). They include polymerase chain reaction (PCR) primers and sequencing adapters. Additionally, a barcode of multiple base pairs (bp) to identify the cell within the droplet and a unique molecular identifier (UMI). And lastly, an oligo(dT) sequence to capture the poly(A) tail of the ribonucleic acid (RNA) and to prime its reverse transcription (Zheng et al. 2017). UMIs consist of multiple bps and are used to label individual RNA molecules as their complementary deoxyribonucleic acid (cDNA) is amplified for sequencing. Thus, with the help of UMIs, we are able to distinguish between copies of one RNA molecule and other transcripts of the same gene (Luecken and Theis 2019, Box 1).

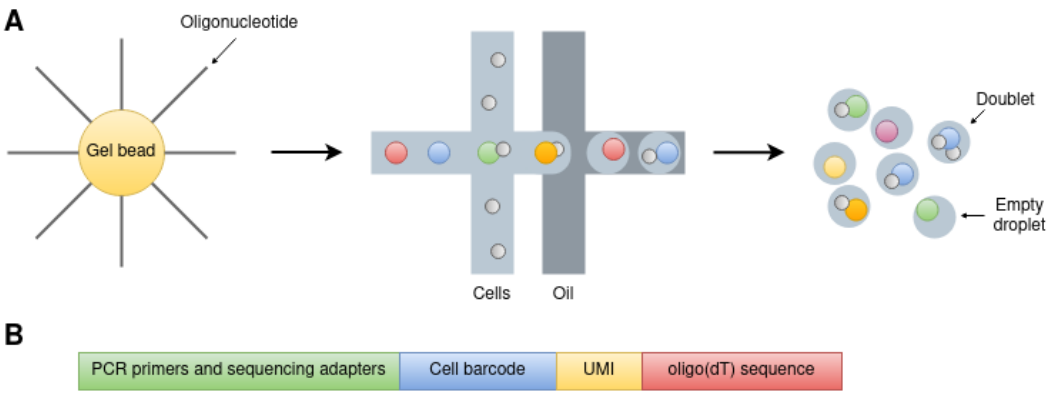


Figure 1: (**A**) Cell isolation with gel beads via oil-encapsulated droplets. (**B**) Components of a gel bead oligonucleotide. Adapted from Figure 1b, d in Zheng et al. 2017 and Figure 1a in AlJanahi, Danielsen, and Dunbar 2018.

Once a cell and gel bead are in a droplet, the cell membrane is ruptured and the present RNA molecules are captured and reverse transcribed. Afterwards, with the emulsion broken, the resulting cDNA molecules are amplified in bulk via PCR and sequenced to produce read data (Luecken and Theis 2019, Box 1; Zheng et al. 2017). This data is then processed by a pipeline called *Cell Ranger*, where the reads are aligned to a reference genome and the corresponding UMIs are assigned to their barcode and a gene (Zheng et al. 2017, Supplementary Methods and Fig. 1f). Each match results in a count. The output of this pipeline is a count matrix that contains the molecular counts for each cell barcode and is used as the input for our scRNA-seq analysis workflows:

	Barcode 1	Barcode 2	...	Barcode N
Gene 1	0	2	...	0
Gene 2	4	0	...	1
⋮	⋮	⋮	⋱	⋮
Gene M	0	0	...	3

Table 1: Example sparse count matrix (*genes* × *barcodes*).

The matrix is sparse due to the low RNA capture efficiency in scRNA-seq. According to AlJanahi, Danielsen, and Dunbar 2018, Table 1, Chromium 10x is able to capture about 14 percent of the cells transcripts. These *dropout events* also account for a large portion of the zeros in the count matrix and occur due to sampling effects in the scRNA-seq workflow described above (AlJanahi, Danielsen, and Dunbar 2018).

Another confounding factor that can occur during scRNA-seq are droplets that capture either none or multiple cells at once. While empty droplets are more common (Chromium has a cell capture rate of approximately 50 percent (Zheng et al. 2017)), they have little impact on the sequencing results. Droplets with multiple cells, also referred to as *multiplets* or *doublets*, are less frequent, but are more difficult to detect and can distort the counts of some barcodes (Luecken and Theis 2019).

2.2 Standard Analysis Workflow

The standard analysis workflow of scRNA-seq data can be divided into two phases, as shown in Luecken and Theis 2019, Figure 1. The first part includes the preprocessing in which the data is prepared and corrected for the upcoming analysis in the second part.

At the beginning of the preprocessing phase, we filter out unreliable cell barcodes from the data by performing quality control (QC). Cell QC is usually done based on three QC metrics (AlJanahi, Danielsen, and Dunbar 2018; Luecken and Theis 2019). The first one is the number of counted transcripts per barcode, or count depth. For example, a barcode with low count depth often indicates that the captured cell is a dying one or that its cell membrane has been broken before capture. An unexpectedly high number of counts for a cell, however, can be

due to doublets or multiplets and must also be removed from the data. The second cell QC metric is the amount of genes expressed per barcode. Because of dropout events, a barcode can contain too few genes, making it unreliable. Finally, the third QC metric is the percentage of mitochondrial reads in a cell barcode, where a high percentage is an indicator of cell stress or a broken cell membrane (AlJanahi, Danielsen, and Dunbar 2018). Therefore, to filter out outliers and faulty barcodes, we need to set lower and/or upper thresholds for the three metrics described. These thresholds must be fitted to the data to avoid filtering cell populations whose unusual characteristics are due to a biological context and not to technical confounders, such as a high number of counts in large cells that could be mistaken for doublets (Luecken and Theis 2019). Tools designed for doublet detection are particularly suitable for this case. In addition to filtering cells, we can also remove genes from the data that are not represented in more than a few cells or from which only a small number of transcripts were measured. Due to the high dropout rate, this usually significantly reduces the number of genes and speeds up further calculations. However, genes that could further differentiate the cells might get lost as a result (AlJanahi, Danielsen, and Dunbar 2018).

The next step is to normalize the data to account for dropout events and bring all cells to a comparable scale. A simple but commonly used method for this is count depth scaling. The counts are divided by the count depth of the corresponding cell and multiplied by a power of 10 (Townes et al. 2019). Afterwards, the data is usually $\log(x+1)$-transformed. This has the advantage that we can measure the changes in gene expressions between cells with log fold changes (Luecken and Theis 2019). The data may also be corrected for batch effects between multiple samples.

To decrease the computational effort in downstream analysis and to visualize the data, we reduce the dimensionality of the count matrix. We start with feature selection, which is the selection of genes for further processing that are highly differentially expressed between cells, usually between 1 000 and 5 000 of them (Luecken and Theis 2019). These are referred to as highly variable or overdispersed genes. Afterwards, the data is further reduced, often with principal component analysis (PCA), a linear approach also used by both pipelines in this thesis. PCA manages to summarize the information of similar genes into a principal component (PC). Each PC captures a portion of the variance in the data, with the first PC having the largest standard deviation. The standard deviation decreases with each subsequent PC, making it appropriate to use only the first n PCs for the following analysis (AlJanahi, Danielsen, and Dunbar 2018). We can choose the number n using an elbow Plot, as shown in Figure 2. After the "elbow" we see that the standard deviation stagnates, so we set the cutoff at 12, since the subsequent PCs contribute little to the variance of the data, and we can thus decrease the computational effort without losing much information. Non-linear methods for visualization continue to reduce the data so that it can be viewed in two-dimensional space. These include, for example, t-distributed stochastic neighbor embedding (t-SNE) (Maaten and Hinton 2008) or uniform manifold approximation and projection (UMAP) (McInnes, Healy, and Melville 2020).

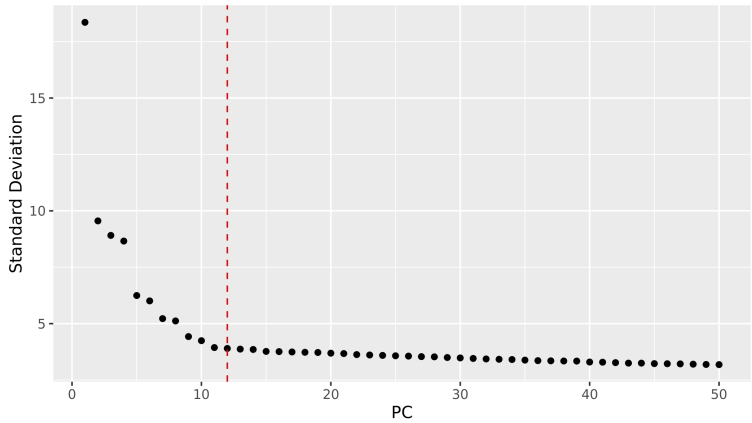
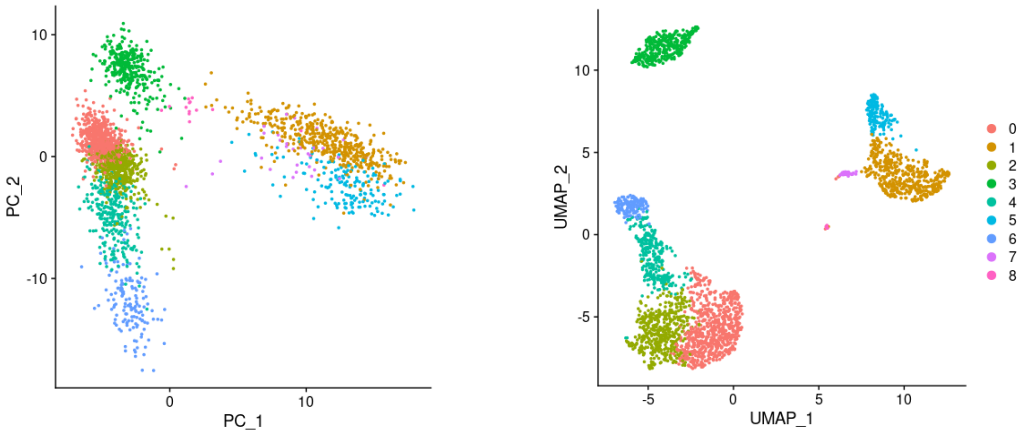


Figure 2: Elbow plot generated by the Pagoda2-pipeline showing the standard deviation of each calculated PC. The higher the standard deviation of a PC, the more information it contains about the variance of the data. The red dashed line indicates the cutoff.

We use PCA to summarize the data because its linearity preserves the actual distances between cells, which is beneficial in downstream analysis. However, in the visualization of the data in two dimensions, this has the effect that the variability between groups cannot be represented ideally. Non-linear methods manage to highlight the local structures in two-dimensional space, but often exaggerate the differences between groups (Figure 3) (Luecken and Theis 2019).



(a) Two-dimensional PCA embedding (linear method).

(b) Two-dimensional UMAP embedding (non-linear method).

Figure 3: Comparison of linear vs. non-linear dimensionality reduction methods in two-dimensional space. The clustering was generated with community detection on the PC-reduced PBMC 3k data set (see Subsection 4.1). We can see that UMAP displays clusters, or local structures, more densely and groups similar ones together. In contrast, in the embedding of the first two PCs, the clusters blend into each other.

Now we move on to the analysis phase, where the options depend heavily on the tool selected for analyzing the scRNA-seq data. We will focus on the methods used in both pipelines, that is, clustering and differential expression analysis.

To analyze the data at the cell level, we group them into clusters. The approach used by the two pipelines in this thesis is community detection. For this purpose, a k-nearest neighbor

graph (KNN graph) is created based on the PC-reduced data, where each cell is treated as a vertex and connected to its k most similar adjacent cells. Densely connected regions of the graph indicate that the cells located there are alike and therefore can be grouped into a cluster (Luecken and Theis 2019). Luecken and Theis (2019) recommend the use of the Louvain algorithm (Blondel et al. 2008) to identify these dense regions. The idea behind this method is to optimize the modularity in the KNN graph. High modularity indicates that vertices within communities are densely connected, while vertices between different communities are only sparsely connected. The Louvain algorithm first searches for small local communities in the graph and then groups them into single nodes. These grouped nodes then form new communities and are merged again. This step is repeated until the maximum modularity in the graph is reached. The final communities then form the cell clusters.

To find out how two groups of cells differ in their gene expression we perform differential expression analysis between them. The two groups can be, for example, cells from one cluster and all other cells in the data. Both of our analysis pipelines use Wilcoxon rank sum tests to do so (Wilcoxon 1945). This non-parametric test determines whether the expression of a gene tends to be higher or lower in one group than in the other. The null hypothesis would be that a gene is expressed equally in both groups. In turn, the alternative hypothesis is that a gene is differentially expressed in the two groups. For this, all counts for a given gene are pooled and assigned ranks. Ranks represent the position of the values in the order of the pooled counts, for example, the counts [8, 5, 10, 2, 13] would get the ranks [3, 2, 4, 1, 5]. Afterwards, the ranks for both groups are summed up respectively and the z-score for the smaller rank sum (W) is calculated, as the large sample size of the cells allows us to use normal approximation:

$$\mathrm{z} = \frac{\mathrm{W} - \mu_{\mathrm{W}}}{\sigma_{\mathrm{W}}}.$$

Here, μ_{W} is the mean and σ_{W} the standard deviation of the rank sums, with n_1 being the sample size of the group with the smaller rank sum and n_2 that of the group with the larger rank sum:

$$\mu_{\mathrm{W}} = \frac{\mathrm{n}_1(\mathrm{n}_1 + \mathrm{n}_2 + 1)}{2}, \quad \sigma_{\mathrm{W}} = \sqrt{\frac{\mathrm{n}_1\mathrm{n}_2(\mathrm{n}_1 + \mathrm{n}_2 + 1)}{12}}.$$

The calculated z-score leads us to the p-value, which we can use to decide whether or not to reject the null hypothesis based on the significance level α. The significance level is typically set to 0.05. If the p-value is less or equal to α, we can reject the null hypothesis, meaning that the gene is differentially expressed between the two groups. Conversely, a gene is equally expressed in both groups if the p-value is greater than α and therefore we cannot reject the null hypothesis.

3 Single-Cell RNA-Sequencing Analysis Pipelines

In this section, we introduce the two scRNA-seq analysis pipelines that we will compare to each other in Section 4. All upcoming figures produced by the pipelines involve the PBMC 10k data set described in Subsection 4.1.

The source code for the Pagoda2-pipeline is available at https://git.hhu.de/pamei104/bsc-projekt-scrna-pagoda2-snakemake. A slightly modified version of the Seurat-pipeline can be found here: https://git.hhu.de/myhuy100/wggc-single-cell. Both links were last accessed on 9th May, 2021.

3.1 Tools Used to Setup and Build the Pipelines

3.1.1 Conda

Conda (*Anaconda Software Distribution* 2016) is an open-source package and environment management system that creates self-contained environments and allows us to easily switch between them. After creating an environment, we can search for, install and update various packages and their dependencies via the command-line. We are also able to define them in advance using an `environment.yaml` file. This way, Conda installs the specified software when building the environment.

By using the latter approach, we get to create predefined environments for each pipeline to ensure reproducibility in terms of packages and an easy setup. Packages that are not available through Conda have to be installed with the R scripts included in the pipelines' repositories.

3.1.2 Snakemake

Snakemake (Mölder et al. 2021) is a workflow management system that we use in order to implement the analysis of scRNA-seq data in form of an automated pipeline. For this, we split our analysis workflow into steps, referred to as *rules*, that are listed in a *Snakefile*. A rule consists of a name and a set of directives. Although not mandatory, most rules have an `input`-directive and an `output`-directive specifying the names of input- and output files, and instructions on how to create the latter from the former. These instructions can be, for example, shell commands, external scripts or regular Python code (Van Rossum and Drake 2009), as Snakemake itself is Python-based. Since both pipelines rely on R packages, we mostly use the `script`-directive here. Within the R scripts, a `snakemake` object is available, through which we can access the rules properties, like the input-, output- or params-directive. We also get access to values in the *config file*, or `config.yaml`, where we can specify data and working directories or parameters used in the workflow with key-value pairs.

By using wildcards, rules can be made generic, as it allows Snakemake to name files and set parameters dynamically. In our case, both pipelines use the config file to fill those wildcards with values.

Furthermore, we can link rules together by specifying the output of one rule as the input of another. Snakemake then automatically determines rule dependencies by substituting all existing wildcards and mapping input files to output files, creating a directed acyclic graph (DAG) of *jobs*. The DAG indicates the order of the jobs and finds out whether jobs that are independent of each other can be executed in parallel, as long as the required resources are available. To connect the rules in our respective workflows, both pipelines save the scRNA-seq data at the end of each rule as a single R object (`.rds` file) and pass it on to the next one for further processing. An exception to this are rules with scripts that generate various outputs from the data, but do not modify the object itself. In this case, the data is read but not stored again, in order to save time and storage space.

Additionally, in cases where the number of output files is unknown before execution, we can use *checkpoints* instead of rules. After each succesful job derived from a checkpoint, Snakemake re-evaluates the job dependencies during workflow execution, because the newly created files have not yet been included in the DAG. The output files of these jobs can then be accessed through a `checkpoint` object in an input function and be returned to a rule, or rather job, depending on them as input. An example of this would be cluster specific files where we want to include the cluster ID in the file name. Since we do not know how many clusters will be created from the data prior to workflow execution, Snakemake cannot determine and replace the wildcards of the cluster IDs when creating the DAG at startup. Therefore, we generate and name the files with their IDs using a script in a checkpoint. With the help of the `checkpoint` object, we then access the files in an input function and extract the cluster IDs from the file names. During the re-evaluation of the DAG, Snakemake can now substitute the wildcards of the cluster IDs with the extracted ones and provide the complete file names as input to a target rule.

3.2 Pagoda2-Pipeline

The following Snakemake scRNA-seq analysis pipeline relies on the R package *Pagoda2* (Barkas et al. 2021), a tool designed to analyze large-scale, single-condition scRNA-seq data sets. Pagoda2 is being developed by the Kharchenko Lab and is partly based on their other software: SCDE (Kharchenko, Silberstein, and Scadden 2014) and PAGODA1 (Fan et al. 2016). In terms of preprocessing, Pagoda2 offers methods for quality control, normalization, feature selection and dimensionality reduction. Further, for downstream analysis, it performs clustering, visualization, differential expression and pathway overdispersion analysis. As a faster alternative to the latter, the analysis tool offers hierarchical differential expression. Pagoda2 also provides an interactive web application where users can further explore the analysis results, such as the clustering or other aspects of heterogeneity in the cells, or perform differential expression analysis between user-defined selections.

The Snakemake DAG of this pipeline (Figure 4) shows that the workflow is linear up to the last step of the analysis. Only then it is possible to run several jobs in parallel. In addition, the pipeline is designed to process only one scRNA-seq data set at a time, as its analysis depends on user-specified parameters that need to be adjusted to the given data, for example, QC thresholds. These parameters are set in advance in the config file, `config.yaml`, although some of them, such as the minimum number of genes per cell, can also be added later. In these cases, the pipeline offers plots from which the user can derive appropriate values.

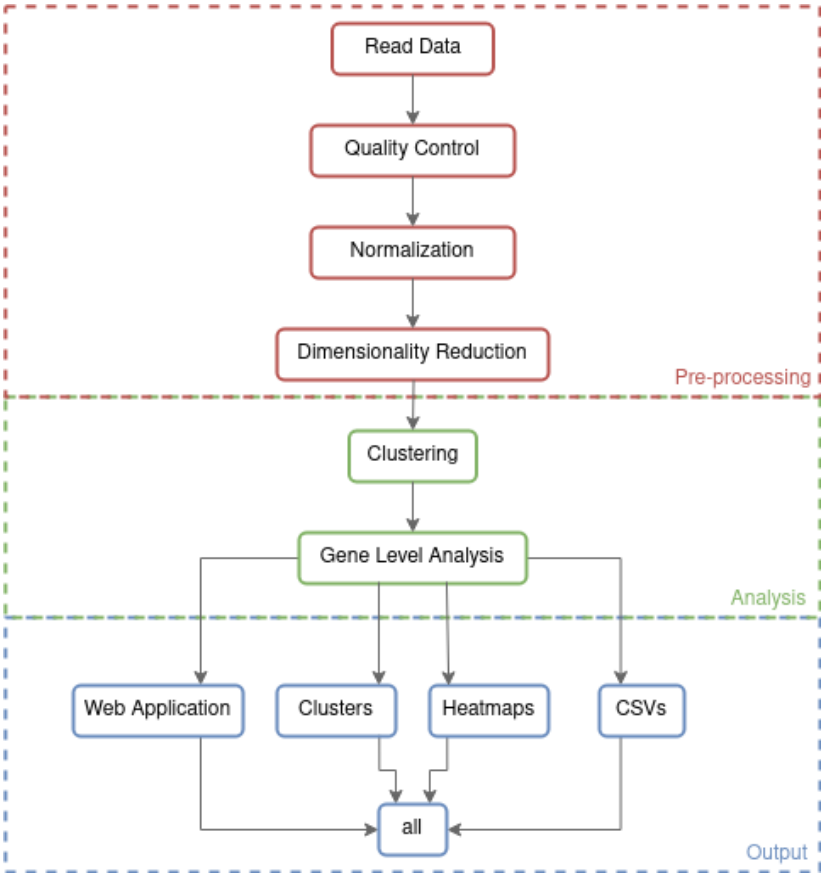


Figure 4: Snakemake DAG of the Pagoda2-pipeline.

3.2.1 Preprocessing

Read the Data The scRNA-seq analysis pipeline starts by reading the count matrix that is located in the directory specified in the configuration file. For this, we use the read-in function for 10x Cell Ranger count matrices that is provided by Pagoda2 and thus obtain the scRNA-seq data as a sparse matrix. Then, the matrix is passed on to the next rule, where we perform QC to filter out unreliable cells.

Quality Control At this point in the workflow, we can set lower and upper bounds on the count depth, a minimum number of genes per cell and an upper limit on the percentage of mitochondrial reads. In order to give an overview of the data, the pipeline provides three different plots, each illustrating one of the three cell QC metrics. We have the option to specify the QC thresholds in the config file beforehand or, with the help of the given plots, add them later. By leaving the respective keys in the `config.yaml` empty, we are prompted to view the generated plots and decide on suitable values while the Snakemake workflow stops in the meantime. These values are also added to the corresponding plots and a summary (Figure 5), in form of a red dashed line, to indicate the thresholding choices.

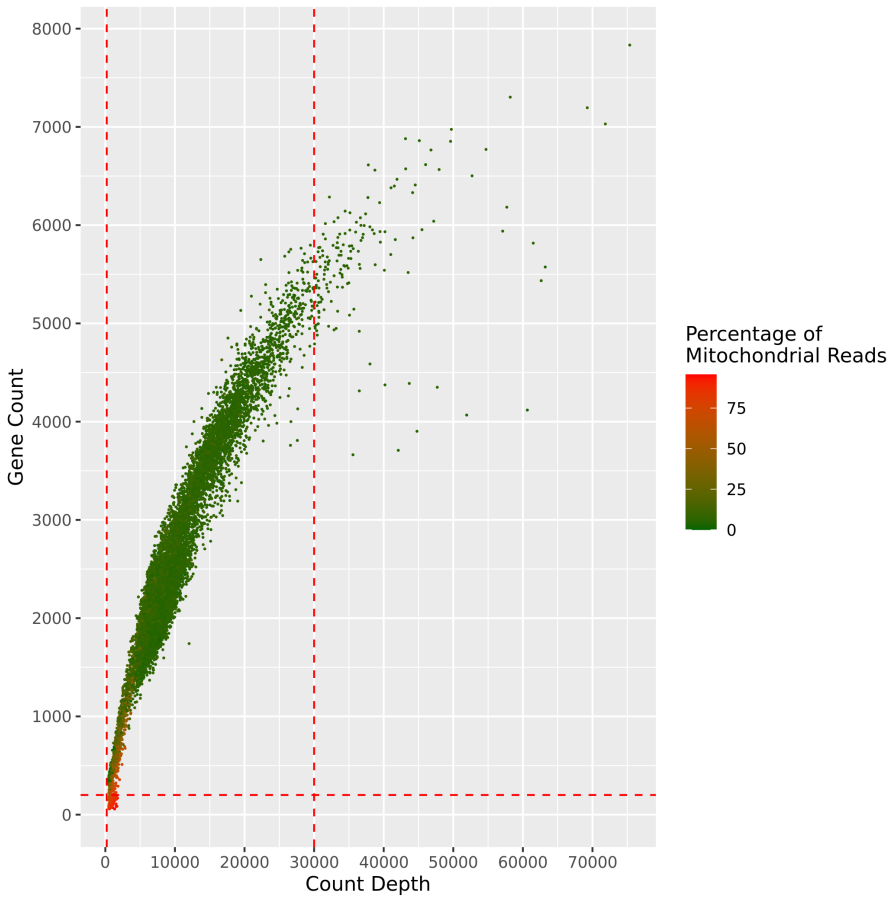


Figure 5: Scatter plot summarizing the filtering decisions made by the user.

In addition, Pagoda2 filters all cells that do not match the expected gene-versus-count relationship, by fitting a linear model over said relationship and removing any outliers from the data (Figure 6).

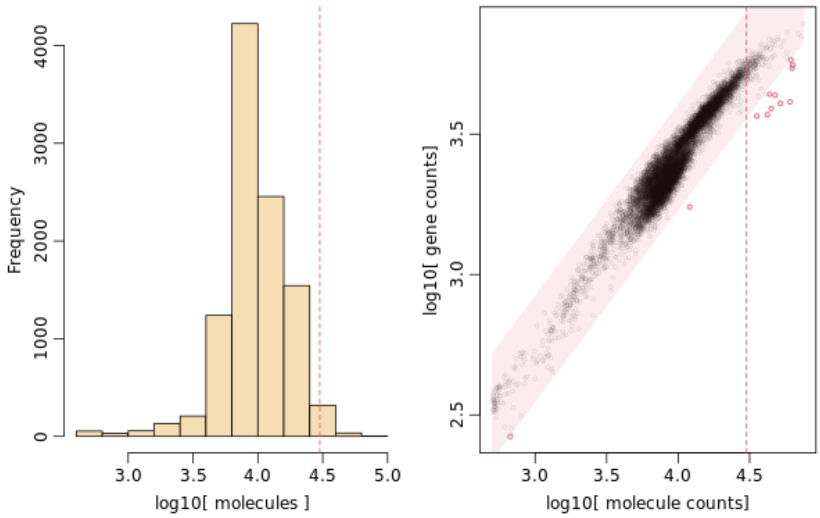


Figure 6: Plots generated by the `gene.vs.molecule.cell.filter` function of Pagoda2. **Left**: Histogram of count depth per cell. **Right**: Gene-versus-count relationship. Red circles represent outliers that are removed from the data.

Finally, the pipeline removes genes that do not exceed the user-specified minimum number of counts across all cells.

Normalization In the following rule the data is normalized by creating a Pagoda2 object that contains the filtered count matrix. From this point on, this object is used for the rest of the preprocessing and the subsequent analysis. When creating the object, the matrix is stored transposed and its counts are normalized with count depth scaling, as described in Subsection 2.2, with a factor of 1 000. Additionally, we get to decide whether to $\log(x+1)$-transform the data or not. Pagoda2 also offers to correct for batch effects, but since this analysis tool is intended to process standalone data sets, its batch correction methods are not optimized, according to the developers. Although the option remains in the pipeline, we do not recommend using it.

Furthermore, we use the `adjustVariance` function of Pagoda2 to normalize the variance of genes in the data. This is because highly expressed genes also show higher expression variance compared to lowly expressed genes, which may not necessarily be due to a subpopulation in the cells, but rather to sampling effects. Therefore, Pagoda2 fits a generalized additive model (Wood 2017) over the variance-versus-magnitude relationship of the genes and rescales their variance to put them on a more comparable scale. By doing so, Pagoda2 also detects overdispersed, or highly variable, genes that are later used in feature selection (Figure 7).

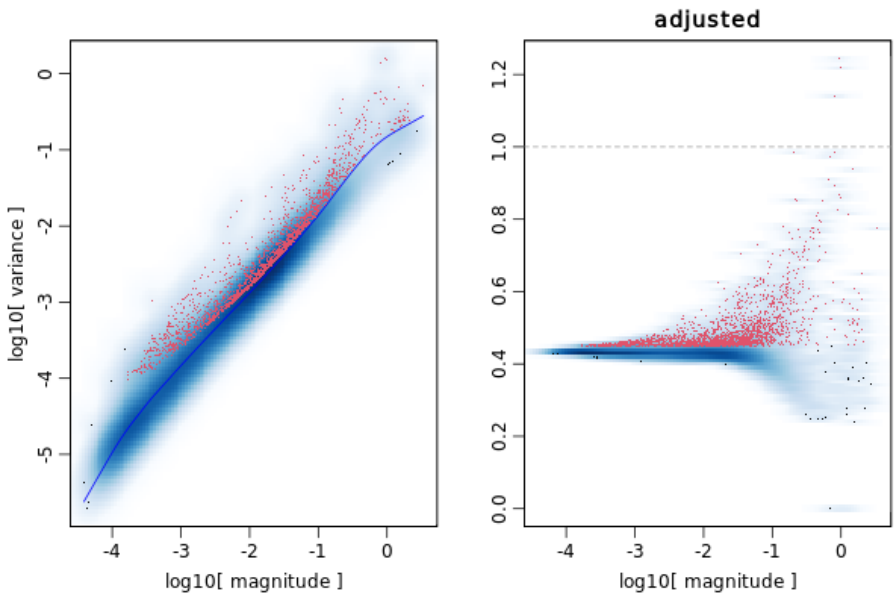


Figure 7: Plots generated by the `adjustVariance` function of Pagoda2. **Left**: Variance-vs-magnitude relationship with fitted smooth linear model. **Right**: Adjusted variance of genes by magnitude. Red dots indicate overdispersed/highly variable genes.

Dimensionality Reduction The last step in the preprocessing portion of this pipeline is to reduce the dimensionality of the data. First, we select, via the `config.yaml`, how many overdispersed/highly variable genes to keep in feature selection. If the number of genes exceeds that of those previously detected in the normalization, Pagoda2 selects additional genes by sorting them by their variance and picking the next ones in order. We then perform PCA on the feature selected data. And just as with the QC thresholds, we again have the option to specify the number of PCs to be used in the upcoming analysis in advance in the config file or add them later. For this purpose, an elbow plot is generated for visualization, showing the standard deviation of the first 50 PCs (Figure 2), which we can use to decide on an appropriate cutoff. After adding the value to the config file, the pipeline adjusts the Pagoda2 object and a red dashed line is added to the plot, indicating our decision. The Pagoda2 object with the reduced data is then passed on to the next rule for data analysis.

3.2.2 Data Analysis

Clustering The analysis of the scRNA-seq data in this pipeline starts with the clustering of the data using community detection. First, the KNN graph is constructed. We set the `k` parameter for graph construction in the config file and can therefore roughly control how many clusters are generated: The larger the `k` is, the less clusters are found by the community detection method. Next, community detection is used on the KNN graph to determine the clusters. The method applied by Pagoda2 is also specified by us in the `config.yaml` and is provided by the R package `igraph` (Csardi and Nepusz 2006). Among them is the `multilevel`-function, which uses the Louvain algorithm recommended by Luecken and Theis (2019).

To visualize the clusters, Pagoda2 now generates two-dimensional embeddings of the data, based on our selections made in the config file. Options include, the often used in visualization, t-SNE, but also UMAP, the Fruchterman-Reingold algorithm (Fruchterman and Reingold 1991) and LargeVis (Liu et al. 2016).

In addition, Pagoda2 constructs a gene KNN graph for the *Show related genes* feature of the web application. This feature allows users to search for genes that resemble each other based on their expression across all cells.

Gene Level Analysis In this step, the clustered data will be analyzed by Pagoda2 at the gene level. To start, Pagoda2 determines the differentially expressed genes of each cluster relative to all others using Wilcoxon rank sum tests. Then, the pipeline prepares a Gene Ontology (GO) (Ashburner et al. 2000; Consortium 2020) environment to collect relevant gene sets for investigation in the web app and for potential pathway overdispersion analysis. Gene sets are considered relevant even if only a part of the corresponding genes are present in the data. To ensure that the correct database is accessed, we need to specify the species from which the scRNA-seq data originates in the config file. The pipeline is able to create GO environments for humans and for mice.

Now, depending on our choice in the config file, the pipeline performs pathway overdispersion analysis or hierarchical differential expression. For pathway overdispersion analysis, Pagoda2 uses a modified version of the method from its predecessor, described by Fan et al. (2016). During this analysis, valid gene sets from the GO environment are tested for coordinated variability in the expression of their genes in the cells. A gene set is considered valid if its number of associated genes is within the pathway size that we specify in the config file, that is, the minimum and maximum number of genes per gene set. Coordinated variability in a gene set exists, for example, when one group of cells highly expresses the associated genes while another group downregulates them. For this, the first PC of the gene set is used to summarize the heterogeneity in gene expression. If the variance in the first PC is higher than expected, the respective gene set is classified as overdispersed and is retained. Since many gene sets share genes, similar sets are subsequently clustered into *aspects* of heterogeneity. To further reduce redundancy, gene sets that show a similar pattern of expression in the cells are also grouped together. The results can then be examined in the web app (Figure 16).

As a faster alternative to pathway overdispersion analysis, Pagoda2 provides a method to perform hierarchical differential expression. In this method, the clusters already generated are further combined with hierarchical clustering, and at each split of the resulting dendrogram, differential expression analysis is performed between the two branches. In the web application, each split represents an aspect of heterogeneity.

The results of the data analysis are then passed on to the remaining rules in the pipeline, which generate various outputs from them.

3.2.3 Output

In addition to the plots from preprocessing and the `.rds` intermediates of the scRNA-seq data, the pipeline generates further output from the data analysis. To visualize the clusterings, the pipeline prints two plots for each selected embedding in the config file, using the `plotEmbedding`-function of the Pagoda2 object. In one, the clusters are labeled with their cluster ID (Figure 13b) and in the other with the corresponding top marker calculated during differential expression analysis. Also, for each cluster, the pipeline provides Comma-separated values (CSV) files of the differential expression analysis results, showing the z-score and the log2 fold change of each included gene. In addition, the CSV files contain the proportion of cells in the cluster expressing a given gene and a boolean indicating whether the expression of a gene is higher than in the other groups. For further visualization of the differentially expressed genes, heat maps are created for each cluster, showing the expression of their respective top 15 markers across all cells. Since we do not know the number of clusters in advance, we use checkpoints instead of rules in Snakemake to integrate the heat maps and CSV files, with cluster IDs in their names, into the workflow.

To provide the web application of Pagoda2 (Figure 16), the pipeline offers two ways to access it. Since the online version of this app needs a running R session, which is closed after the Snakemake workflow is finished, the pipeline saves the app as an `.rds` file. Additionally, it creates an R script, that can be started manually within the Conda environment and loads the app into an active R session. The more convenient alternative is to load the application's binary file (`.bin`), also generated by the pipeline, directly into the offline app in the browser. This way, no active R session is needed, however, this offline version of the app is slower than its online counterpart.

3.3 Seurat-Pipeline

The Snakemake scRNA-seq analysis pipeline presented in this subsection uses the R package *Seurat* (Hao et al. 2020) as its main component, developed by the Satija Lab. This scRNA-seq analysis tool can apply the standard preprocessing procedures, cluster data and perform differential expression analysis. In contrast to Pagoda2, we can process and integrate data sets with multiple conditions using Seurat. For example, we can use this pipeline to compare diseased and healthy tissues and look for differences in gene expression.

For reliable doublet detection, the QC includes the R package *DoubletFinder* (McGinnis, Murrow, and Gartner 2019), which detects and removes doublets from scRNA-seq data. McGinnis, Murrow, and Gartner (2019) have shown that this procedure improves differential gene expression analysis as it results in newly identified differentially expressed genes, which would otherwise be obscured by doublets.

The structure of the Snakemake DAG of this pipeline, shown in Figure 8, is similar to that of the Pagoda2-pipeline (Figure 4). The rules up to the marker discovery cannot be parallelized either. However, preprocessing involves data integration, and the analysis and output parts overlap. This pipeline also processes only one data set at a time, due to the parameters that need to be adjusted to the data, such as the expected percentage of doublets.

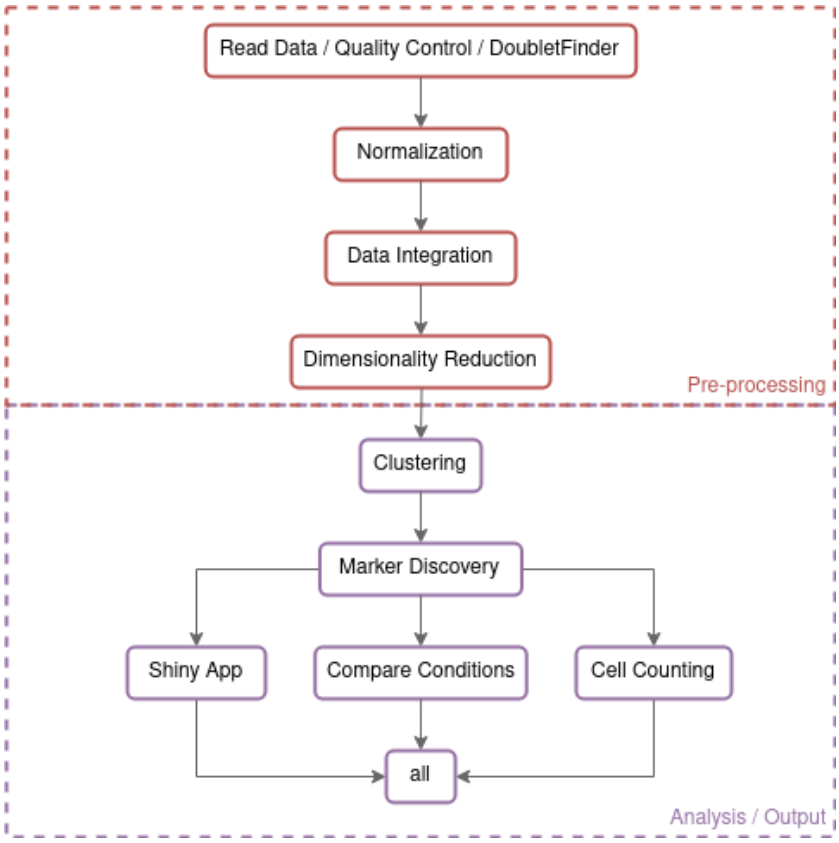


Figure 8: Compressed Snakemake DAG of the Seurat-pipeline.

3.3.1 Preprocessing

Quality Control The first step in this scRNA-seq analysis pipeline is to load the count matrix with Seurat's read-in function, creating a Seurat object. During this process, cells expressing less than 200 genes are removed from the data, as well as genes that are only present in less than three cells. Afterwards, the data is split into its samples for further QC, if possible. These samples could, for example, each contain a batch from the same sequencing experiment or consist of cells from different experiments. The result is a list of Seurat objects with one sample each, which is passed on to the next rule.

The maximum percentage of mitochondrial reads per sample is set by us in the config file. Barcodes that exceed the cutoffs are subsequently filtered. For visualization, two plots are generated for each sample showing the three cell QC metrics from Subsection 2.2, before and after filtering by the cutoffs (plot from after filtering the cells: Figure 9). If the keys for the thresholds in the `config.yaml` are left empty, we may view the plots of the unfiltered cells, decide on appropriate cutoffs and add them to the config file.

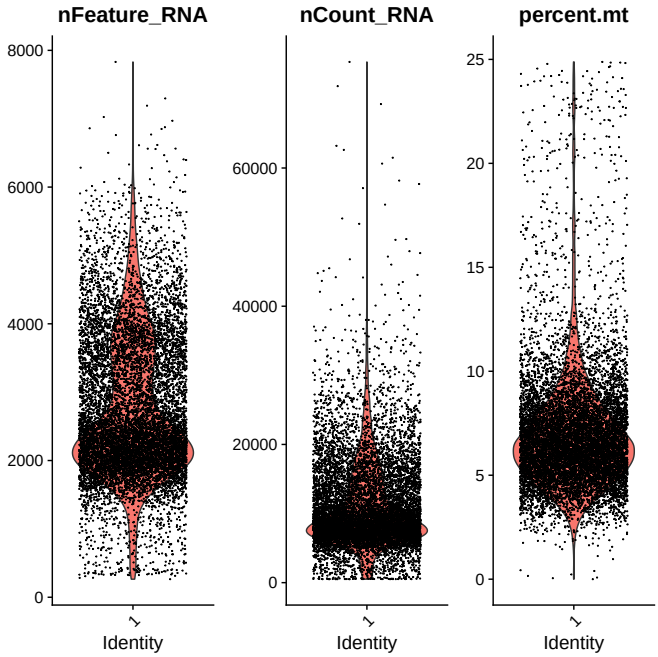


Figure 9: Violin plots of three QC metrics: Number of features/genes per cell, number of counts per cell and the percentage of mitochondrial reads per cell. Measured **after** filtering cells with high mitochondrial read percentage.

The next step in the QC of this pipeline is to perform doublet detection with DoubletFinder. First, we specify the expected percentage of doublets per sample in the config file. The expected number of doublets varies with the size of the sample: The more barcodes in a sample, the greater the rate of doublets in it. DoubletFinder requires a fully preprocessed Seurat object to work. For this, the pipeline applies the `SCTransform`-function (Hafemeister and Satija 2019) to each sample, which covers all necessary steps, such as normalization and feature selection. Lastly, PCA and UMAP is used to reduce the dimensionality of each sample. To assist in select-

ing the number of PCs to use, the pipeline generates elbow plots for each sample, from which we can derive the appropriate cutoffs and enter them into the config file.

We will now summarize the DoubletFinder workflow as described in (McGinnis, Murrow, and Gartner 2019). DoubletFinder starts by generating artificial doublets from the available data. For this reason, multiple samples are treated individually, so that no doublets are created from cells that do not originate from the same batch and thus cannot exist as a real doublet. Then, DoubletFinder merges the artificial doublets with the real data and preprocesses them again. To infer the similarity between simulated doublets and potentially real ones, DoubletFinder uses PCA and the resulting PC distance matrix to calculate the proportion of artificial k-nearest neighbors (pANN) for each cell. A high proportion means that the cell resembles the simulated doublets, which is presumably because the cell itself is one. Finally, these pANN values are sorted and cells with the top x values are removed from the data. Here, x is limited to the expected percentage of doublets per sample. To show which cells were removed by DoubletFinder, the pipeline creates UMAP plots with the classified cells (Figure 10).

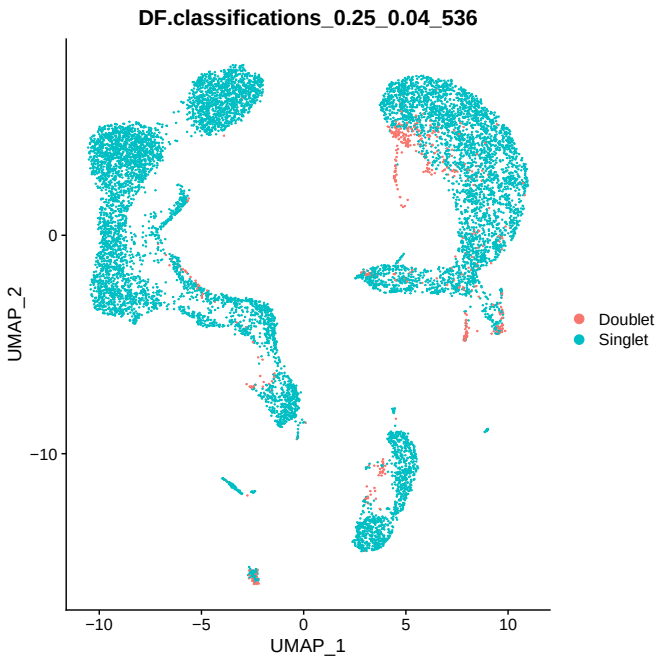


Figure 10: UMAP embedding with classified doublets (red) and singlets (blue) in the data.

Normalization After adding user-defined metadata to the scRNA-seq data, such as conditions for later comparison, the data is prepared for dimensionality reduction and possibly data integration. To do that, the data, or each sample, needs to be normalized again with SCTransform. In contrast to the commonly applied count depth scaling, SCTransform uses generalized linear models to remove sampling effects, such as dropouts, from the data. The used approach is further described by Hafemeister and Satija (2019). The results of SCTransform are stored in an assay in the Seurat object called *SCT*. Assays in a Seurat object store raw, normalized and scaled data as well as other metadata, and allow us to safe multiple variations of the scRNA-seq data, for example, with different normalization methods applied.

Integration If the scRNA-seq data consists of multiple samples, the pipeline now merges the list of Seurat objects into a single object for later analysis. First, Seurat identifies features, or genes, that are variable across all samples and selects the 3 000 top scoring ones. Using that information, it then searches for cells across all samples that resemble each other and therefore could be of the same cell type. Seurat uses these pairs of cells, referred to as *anchors*, to integrate the samples and correct for technical confounders such as batch effects. The resulting data is stored in the *integrated* assay of the Seurat object. Last, the pipeline reduces the dimensionality of the merged data with PCA. If the scRNA-seq data consists of only one sample, data integration is skipped and only PCA is applied.

Dimensionality Reduction The final rule of the preprocessing phase involves further dimensionality reduction with UMAP. Here, we specify the number of PCs to use for the reduction in the config file and may view the elbow plot, which was printed after data integration, to select an appropriate value. The pipeline then creates another elbow plot with the selected cutoff and a plot illustrating the UMAP embedding.

3.3.2 Data Analysis

Clustering The scRNA-seq data analysis in this pipeline starts with clustering of the cells. First, Seurat constructs a KNN graph on the integrated, dimensionality-reduced data. The `k`-parameter is set to 20 and we decide in the config file on how many PCs to use. Based on the KNN graph, Seurat then creates a shared nearest neighbor graph (SNN graph) by computing the neighborhood overlap of every cell and its 20 nearest neighbors.

Afterwards, community detection is performed on the SNN graph to generate the clusters, using the Louvain Algorithm. We decide with the `resolution`-parameter of Seurat's `FindClusters` function on how large the number of discovered communities should be. The `resolution`-parameter is a numeric value and is set in the `config.yaml`. A value above 1.0 leads to a greater number of communities, while a value below 1.0 results in a smaller amount. We can leave the value for the resolution empty and instead pass a list of various ones to the `choosableResolutions`-key in the config.yaml. The pipeline then returns the clustering for each listed resolution in the form of plots, from which we can choose a fitting value to use.

Finally, the pipeline prints a UMAP embedding of the clustering, annotated with the cluster IDs (Figure 13a). If the data consists of multiple conditions, an additional plot of the clustering is created, in which the cells are split into their respective condition.

Marker Discovery The next step is to look for differentially expressed genes, or marker genes, that characterize each cluster. However, we must first normalize the data again, this time using the `NormalizeData` and `ScaleData`-functions provided by Seurat. Unlike SCTransform, NormalizeData uses count depth scaling with a factor of 10 000 to normalize the count data and $\log(x+1)$-transforms it afterwards. The results are saved to the *RNA* assay of the

Seurat object, following the recommendation of the Seurat developers to run differential expression on said RNA assay. Next, Seurat calculates the differentially expressed genes by comparing each cluster against all others, using Wilcoxon rank sum tests. The results of all clusters are stored in a CSV file. This file includes the p-values and log2 fold change of the genes, as well as the proportion of cells in the cluster expressing each gene and, for comparison, the proportion in all other clusters. In addition, the pipeline provides a CSV file containing the average expression of each gene across all clusters.

Compare Conditions If the scRNA-seq data contains two conditions, the pipeline calculates their differences in gene expression for all clusters, again using Wilcoxon rank sum tests. The results are also saved to a CSV file, one for each cluster, and show the p-value and log2 fold change of each gene. In addition, the proportion of cells in the cluster expressing a gene is given for both conditions.

3.3.3 Output

In addition to the output generated during preprocessing and data analysis, the scRNA-seq analysis pipeline also creates an interactive web application using the R package *ShinyCell* (Ouyang 2021). ShinyCell provides the automated creation of a web app that visualizes the scRNA-seq data in different ways. For example, we can study the co-expression of genes in cells (Figure 17) or compare cell information between different clusters, samples or conditions. The app can later be started in an active R session.

At the end, the pipeline creates CSV files showing the distribution of samples and, if present, conditions across all clusters.

4 Comparison

To compare the two scRNA-seq analysis pipelines, we ran them independently on the same data. The parameters for which it is possible were chosen similarly, for example, the upper threshold on the percentage of mitochondrial reads in a cell or the number of PCs to be used in downstream analysis.

4.1 Data

The data used in this comparison includes four data sets of different sizes. Three of them are publicly available on the 10x Genomics website (10x Genomics 2021). They contain the count matrices of human peripheral blood mononuclear cells (PBMCs) with about 3 000, 10 000 and 33 000 barcodes respectively (10x Genomics 2016a; 10x Genomics 2020; 10x Genomics 2016b). The fourth data set, here referred to as *Mice 67k*, consists of approximately 67 000 barcodes and has two conditions that can be compared to each other.

In order to measure the runtimes of both pipelines, we ran them on all four data sets mentioned above. For a closer look at the preprocessing and analysis results, we chose the PBMC 10k data set. The reason for this is that Pagoda2, unlike Seurat, is not designed to compare multiple conditions with each other and we will therefore look at the analysis of single-condition data. Furthermore, the PBMC 10k data set is the most recent of the three single-condition 10x Genomics data.

4.2 Runtimes

We now look at the runtimes of each pipeline in Table 2. Each Snakemake execution used twelve cores and was run on the same machine, with 1 TiB DDR4 RAM and an AMD EPYC 7742 64-Core Processor with 128 Threads. For data sets with only one condition, we skipped the rule in the Seurat-pipeline that compares two conditions within each cluster. Note that these times do not represent the respective main analysis tools per se, but the workflows in which they are implemented. We mention this because when processing large data sets, a great amount of time is spent saving and loading the scRNA-seq data as `.rds` files in between Snakemake rules. And except for the initial read-in of the sparse count matrix, the functions to do so are provided by the R base package.

Data set	Conditions	Samples	Barcodes	Runtime (Hr:Min)	
				Pagoda2	Seurat
PBMC 3k	One	One	2 700	00:03	00:10
PBMC 10k	One	One	10 985	00:18	00:45
PBMC 33k	One	Six	33 148	00:29	02:02
Mice 67k	Two	Six	67 338	02:05	06:15

Table 2: Runtimes of both pipelines on data sets with different sizes and conditions. The times were taken from the log files of Snakemake and cover the entire workflow, from reading the data to the final `all` rule.

We can see that the Pagoda2-pipeline is faster than the Seurat-pipeline across all four data sets. This is partly due to the use of DoubletFinder in the Seurat-pipeline. With the exception of the Mice 67k data set, the DoubletFinder portion itself takes at least as long as the entire Pagoda2-workflow, as seen in Figure 11. The large increase in the runtime of DoubletFinder from the PBMC 10k data to PBMC 33k is related to the Seurat-pipeline processing each of the six samples individually in the QC and thus also with DoubletFinder.

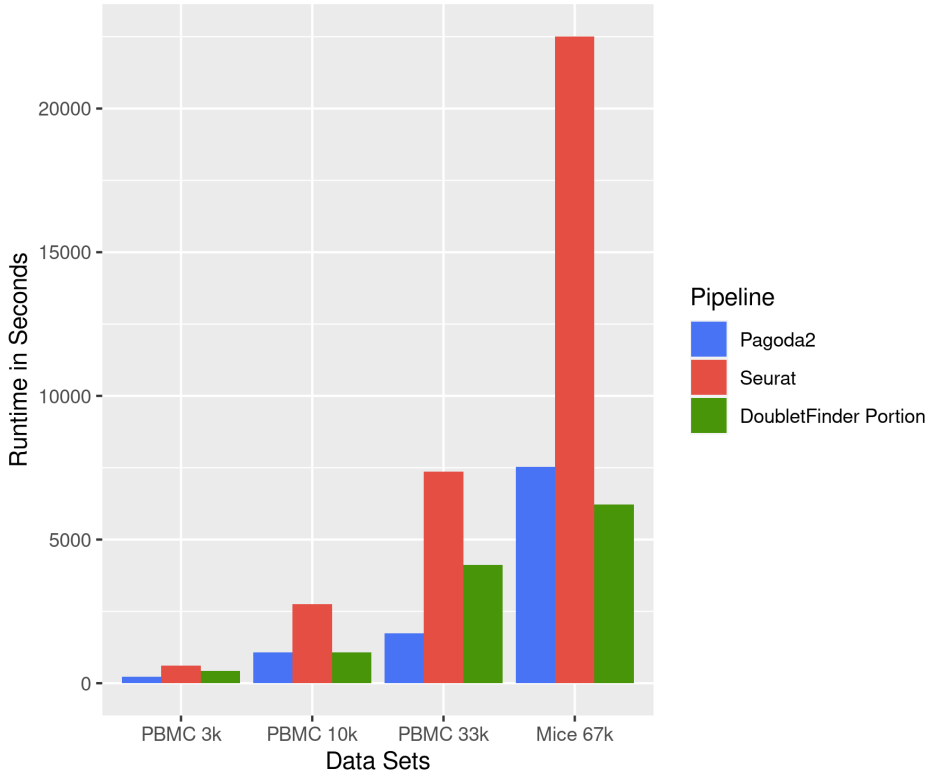


Figure 11: Runtime of each pipeline in seconds, measured on different data set sizes. The values are taken from the log files provided by Snakemake. The blue and red bars represent the Pagoda2 and Seurat pipelines, respectively. The green bars indicate the DoubletFinder portion of the Seurat-pipeline.

Another reason why the Pagoda2-pipeline is faster is the number of rules and the associated storage and loading times of the `.rds` files in between, as mentioned earlier. This is because the Seurat-pipeline creates twelve to thirteen `.rds` files of the scRNA-seq data during execution, depending on whether data with one or two conditions are used, most of which are loaded again in another rule. In contrast, the Pagoda2-pipeline stores seven `.rds` files of the scRNA-seq data including the web app, with the advantage that the Pagoda2 objects require less storage than their Seurat counterparts and thus have generally shorter saving and loading times. Most of the other files, such as the created plots, are too small in size to make a noticeable difference in the runtimes.

The increase in the runtime of Pagoda2 on the Mice 67k data set was most noticeable in the pathway overdispersion analysis, which takes much longer with larger data sets. To further reduce computation time, hierarchical differential expression can be used as an alternative, although the results are less informative.

4.3 Analysis Results

The following analysis results are from the two scRNA-seq analysis pipelines that were run independently on the PBMC 10k data set. We will examine them for similarities and differences in QC and clustering. Additionally, we will briefly discuss the results of the pathway overdispersion analysis by Pagoda2.

Quality Control Using Table 3, we can view how many cells and genes are present in the scRNA-seq data before and after the filtering process of the analysis tools.

	Unfiltered	Pagoda2	Seurat/Doubletfinder
Cells	10 985	10 140	9 767
Genes	36 601	19 389	22 860

Table 3: Number of cells and genes, before and after filtering by each pipeline.

Starting with the cells, the Pagoda2-pipeline filtered out 845, while the Seurat-pipeline with DoubletFinder removed 1 218 barcodes from the data. The user-defined thresholds in the Pagoda2-workflow, such as the minimum number of genes and the maximum percentage of mitochondrial reads per barcode, were set to the same values as those used in the Seurat-pipeline, 200 and 25%, respectively. This implies that the difference in the number of cells filtered is due to the way the cell sizes were treated. In the Pagoda2-pipeline, cell sizes were restricted to a minimum of 200 and a maximum of 30 000 counts per cell. Since the raw PBMC 10k data set was trimmed a priori to a minimum size of 500 counts per barcode, the lower bound set is obsolete. This allows us to make a direct comparison between the upper threshold of the Pagoda2 workflow, which is intended to filter doublets, and the DoubletFinder tool, which is designed for this purpose. Figure 12 shows that the cells filtered by the upper bound of the Pagoda2-pipeline only slightly overlap with those that were also classified as doublets by DoubletFinder. Therefore, assuming DoubletFinder is correct in its prediction, many valid cells were removed from the data. DoubletFinder also manages to identify barcodes as doublets that lie between the cell size thresholds of the Pagoda2-pipeline, as shown by the blue circles in Figure 12. These are droplets that could have captured two cells, neither of which have a high number of transcripts and thus are not considered as being too large for a single cell by the upper limit.

The numerous cells, in Figure 12, that have a gene count between 200 and 1 000 and were not retained in either workflow can be explained by the shared upper threshold on the proportion of mitochondrial reads. The isolated red circle, at approximately 12 000 count depth and 1 800 gene count, was considered an outlier from Pagoda2 and therefore removed from the data (Figure 6).

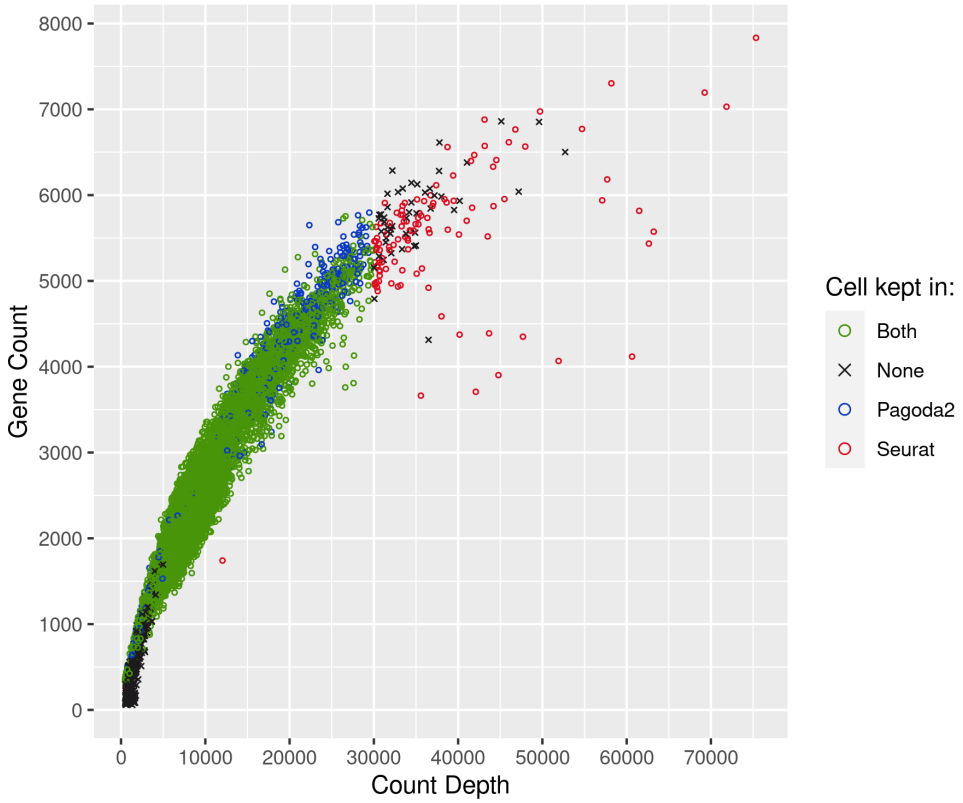
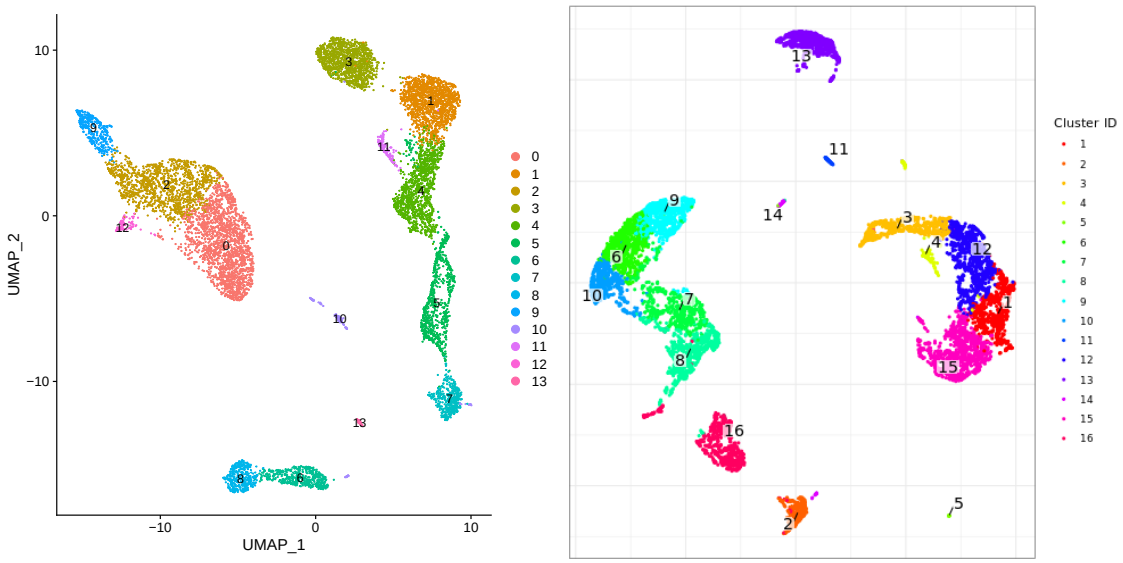


Figure 12: Scatter plot showing how the cells were filtered in the respective pipelines. Each data point represents a cell barcode. A green circle indicates that the barcode was kept in both pipelines. A blue circle represents that the barcode was only kept in the Pagoda2 pipeline, while a red circle represents a barcode that was only allowed in the Seurat pipeline. A black cross is a barcode that was not retained after either QC and thus was not included in any downstream analysis.

In terms of genes, the Pagoda2-pipeline removed 17 212 from the scRNA-seq data, while the Seurat-pipeline filtered out 13 741 genes. This leads us to the assumption that the procedure of filtering genes based on a minimum number of counts, such as in the Pagoda2 workflow, is more strict, than specifying a minimum number of cells in which the genes must be detected, as done by Seurat.

Clustering Our main interest in comparing the two clusterings is whether both scRNA-seq analysis tools, Pagoda2 and Seurat, group the same cells together. Looking at the two plots in Figure 13, we see that Pagoda2 divided the cells of the PBMC 10k data set into 16 clusters (IDs: 1-16), while Seurat identified 14 clusters (IDs: 0-13). For the clustering in each pipeline, we used the same number of PCs for dimensionality reduction and set the k parameters to 20 when constructing the KNN graphs in both workflows. Furthermore, community detection was performed with the recommended Louvain algorithm. Differences in the number of clusters and the distribution of the barcodes could be due to the additional construction of the SNN graph in the Seurat-Pipeline or the varying normalization approaches: count depth scaling in Pagoda2 and Seurat with SCTransform. Another consideration is that the cells that remained only in the Pagoda2-workflow after QC form additional clusters. After all, the Pagoda2-pipeline kept 487

cells that were filtered in the Seurat-pipeline. Conversely, the Seurat-pipeline retained 114 barcodes that its counterpart removed from the data. However, in Tables 5 and 6, we see that no cluster consists of a majority of cells that were retained in only one pipeline. The deviating number of clusters in the Seurat-pipeline is therefore related to the `resolution`-parameter in the `FindCluster` method of Seurat. This parameter was set to 0.5 for this run, which leads to a smaller number of communities compared to higher values: With 0.7 we obtain a clustering with 16 clusters, the same amount as in Pagoda2. A resolution of 1.0 creates 18 clusters.



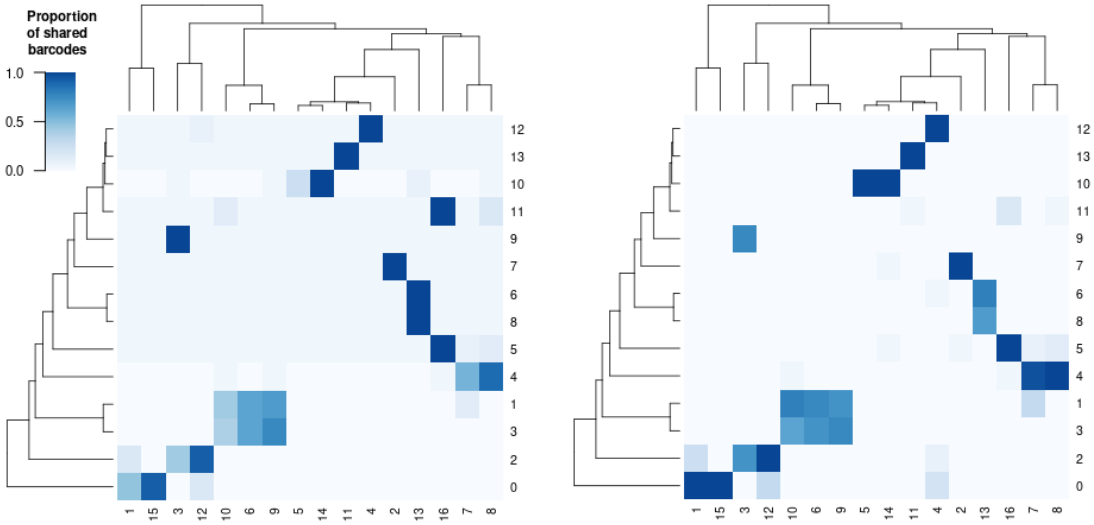
(a) Clustering of the Seurat-Pipeline (`resolution`-parameter set to 0.5).

(b) Clustering of the Pagoda2-Pipeline (without top marker annotation).

Figure 13: UMAP embeddings of the cell clusterings created with the Louvain algorithm.

To determine which groups are similar between the two clusterings, we look at the number of matching barcodes in each case. The heat maps in Figure 14 show us the proportions in which the cells from one clustering map to the other. For the specific numbers, please refer to Table 7.

As we see in Figure 14a, Seurat's clusters 5-13 can each be almost completely assigned to a single Pagoda2 cluster using their matching barcodes. Clusters 0-4, however, distribute their barcodes more evenly across the clusters of Pagoda2. In addition, in the column for cluster 5 of Pagoda2, we see that no cluster of Seurat can be mapped in majority to it. This can be explained by the fact that cluster 5 includes only 20 cells in total, all of which also occur in cluster 10 of Seurat. However, the latter also shares 85 barcodes with cluster 14 of Pagoda2. Similarly, in Figure 14b we notice that no cluster of Pagoda2 maps in majority to cluster 11 of Seurat. The highest match for this group is cluster 16 of Pagoda2 with 104 barcodes, but the majority of cells in cluster 16 can be assigned to cluster 5 of Seurat.



(a) Barcode matches scaled by row.

(b) Barcode matches scaled by column.

Figure 14: Heat maps showing how the shared barcodes are distributed from one clustering to the other. The rows represent Seurat's clusters and the columns those from Pagoda2. The dendrograms were created using hierarchical clustering based on the distance matrix of Table 7 with euclidean distances.
(a) Scaled by row to illustrate the distribution of the Seurat clusters barcodes across the Pagoda2 clusters.
(b) Scaled by column to show the distribution of the Pagoda2 clusters barcodes across the Seurat clusters.

When we compare both heat maps, we see patterns that allow us to assign groups from both clusterings to each other, building structures, which can also be seen in both Figure 13a and Figure 13b. For example, the barcodes from both clusters 6 and 8 of Seurat are all present in cluster 13 of Pagoda2, except for 2 outliers. The UMAP embeddings show that Seurat has grouped the same cells as Pagoda2, but divided them into two subclusters. Most structures in the UMAP embeddings can be assigned to each other as follows, based on their matching barcodes, ignoring outliers:

Seurat clusters	Pagoda2 clusters
0, 2, 9, 12	1, 3, 4, 12, 15
1, 3, 4, 5, 7, 11	2, 6, 7, 8, 9, 10, 16
6, 8	13
13	11

Table 4: Groups of clusters of both pipelines that share barcodes and a similar structure.

Clusters 5 and 14 of Pagoda2 form an exception, as they are clearly separated in their respective UMAP embedding, although almost all of their barcodes are contained in the single cluster 10 of Seurat. Thus, we observe that the two clusterings share similar structures, but these slighty differ in the number of clusters and how strictly these are separated.

To quantify the similarity of the clusterings, we use the Rand index (RI) (Rand 1971). The RI is calculated as follows:

$$\mathrm{RI} = \frac{\mathrm{a} + \mathrm{b}}{\binom{n}{2}}.$$

Where a is the number of pairs that are in the same cluster in both clusterings and b is the number of pairs that are in different clusters in the two clusterings. The denominator is the amount of possible unordered pairings of the cells. The values of the RI range from 0 to 1, where 0 states that both clusterings do not agree on any pair classification and 1 indicates that both clusterings are identical. The RI for the clustering from Pagoda2 and Seurat's clustering with the `resolution`-parameter set to 0.5 is 0.9059. This confirms our observation that the two clusterings are similar. If we raise the resolution parameter to 0.7 to match the number of clusters in both clusterings, we obtain an RI of 0.9164, a slight increase in similarity.

Pathway Overdispersion Analysis Since Seurat does not perform pathway overdispersion analysis and thus we cannot make a comparison, we briefly look at part of the results from the PBMC 10k data set. We can see in the center heat map of the Pagoda2 web application (Figure 16) that seven aspects of heterogeneity were found in the cells by the pathway overdispersion analysis. Orange indicates coordinated upregulation of the gene sets contained in an aspect, while green indicates downregulation. We now take a closer look at aspects 2 and 3 in Figure 15.

(a) Aspect 2. (b) Aspect 3.

Figure 15: UMAP embeddings highlighting aspects of heterogeneity found in pathway overdispersion analysis of the PBMC 10k dataset.

In Figure 15a, aspect 2 is highlighted in the clusters and we notice that the clusters 1, 3, 4, 12 and 15 all upregulate the gene sets in this aspect. In contrast, most other cells appear to downregulate them. Gene sets, or biological processes, that are present in this aspect include: *myeloid leukocyte activation*, *neutrophil activation involved in immune response* and *neutrophil degranulation*. Based on this, we can assume that the cells highly expressing the corresponding genes are granulocytes, or more precisely, neutrophils.

If we examine Figure 15b, we see that cluster 13 upregulates the genes sets in aspect 3. And even though the other cells do not downregulate them, cluster 13 still stands out. Aspect 3 contains the following gene sets/pathways, among others: *B cell receptor signaling pathway*, *regulation of B cell activation* and *immunogoblin complex*. From this, we presume that the cells in cluster 13 are B cells.

5 Discussion

In this section, we discuss the advantages and disadvantages of Snakemake as a workflow tool for scRNA-seq analysis pipelines and draw conclusions from the comparison of the two pipelines.

5.1 Use of Snakemake

Snakemake, as a workflow tool in both scRNA-seq analysis pipelines, has proven useful during development as well as in execution. In development, we keep a good overview of the different steps of an scRNA-seq analysis workflow, since each rule usually represents one of these steps and each associated script performs a single task. In use, Snakemake takes care of saving and naming the many outputs of the pipelines and creates a clear folder structure for each project. Through the configuration file, we can also easily adjust each workflow execution to our liking.

A drawback of using Snakemake are the storage and loading times when transferring the scRNA-seq data between rules. These would not be necessary in a single R script. However, we can reuse the resulting intermediates in repeated or subsequent workflow executions. For example, if we later change the parameters for the clustering, we do not have to repeat all the steps in preprocessing, as these have not changed and are saved in the processed `.rds` file of the scRNA-seq data. Thus, we can save time in later workflow runs of the same project.

5.2 Conclusions

Looking at the DAGs and comparing them using the PBMC 10k data, we see that the scRNA-seq analysis pipelines resemble each other at some points, such as the choice of community detection for clustering or dealing with mitochondrial reads in the data. However, we notice that the Seurat-pipeline can filter the cells more accurately by using DoubletFinder, since it does not need to rely on a fixed upper threshold that might filter out valid cells. With Tables 5 and 6, we saw that it did not have much impact in clustering in our example run. Although, it could lead to better results in the analysis for larger data sets, where the rate of doublets is also higher. The biggest difference in the pipelines is due to their main components, Seurat and Pagoda2, and is the type of data they can process. Since Pagoda2 is designed for standalone data sets, it is not able to efficiently analyze data with multiple conditions. For example, it lacks the ability to integrate multiple data sets, or the existing batch correction methods are not optimized enough to be used, and cannot compare conditions in data, as Seurat does. However, Pagoda2 is able to perform pathway overdispersion analysis and is comparatively faster than the combination of Seurat and Doubletfinder. Therefore, the Seurat-pipeline is the preferred choice when analyzing data sets with multiple conditions and samples. The Pagoda2-pipeline, in turn, can be considered when dealing with single-condition data or when the expression of pathways and gene sets plays a major role in the research question.

5.3 Outlook

As the number of scRNA-seq analysis tools increases and new methods are developed, the two pipelines can be expanded in the future. For example, automatic cluster annotation can be applied to identify cell types more quickly. This involves dedicated tools comparing gene expression profiles from reference databases with cells and annotating them accordingly. Also, methods for trajectory inference help to detect dynamic processes in cells, such as cell differentiation (Luecken and Theis 2019). With Snakemake, the scRNA-seq workflows can even be extended by tools from other programming languages, as long as the option for data conversion is available. In the case of the Pagoda2-pipeline, a doublet detection tool could be added to the QC.

We can reduce the runtimes of the pipelines by using Snakemake's `pipe`-flag for the `.rds` output files of the scRNA-seq tools. This way these objects stay in memory and we don't have to save and load them between each rule. For the Seurat-pipeline, we could also parallelize the processing of each sample by DoubletFinder, saving time in QC. To run the pipelines on a cluster, we would also need to mark the required resources on each rule.

6 References

[1] Luke Zappia, Belinda Phipson, and Alicia Oshlack. "Exploring the single-cell RNA-seq analysis landscape with the scRNA-tools database". en. In: *PLoS Computational Biology* 14.6 (June 2018), e1006245. DOI: 10.1371/journal.pcbi.1006245. URL: https://doi.org/10.1371/journal.pcbi.1006245.

[2] Felix Mölder et al. "Sustainable data analysis with Snakemake [version 1; peer review: 1 approved, 1 approved with reservations]". In: *F1000Research* 10.33 (2021). DOI: 10.12688/f1000research.29032.1.

[3] Nikolas Barkas et al. "pagoda2: Single Cell Analysis and Differential Expression". R package version 1.0.2. 2021.

[4] Yuhan Hao et al. "Integrated analysis of multimodal single-cell data". In: *bioRxiv* (2020). DOI: 10.1101/2020.10.12.335331. URL: https://doi.org/10.1101/2020.10.12.335331.

[5] Fuchou Tang et al. "mRNA-Seq whole-transcriptome analysis of a single cell". In: *Nature Methods* 6.5 (May 2009), pp. 377–382. ISSN: 1548-7105. DOI: 10.1038/nmeth.1315. URL: https://doi.org/10.1038/nmeth.1315.

[6] Byungjin Hwang, Ji Hyun Lee, and Duhee Bang. "Single-cell RNA sequencing technologies and bioinformatics pipelines". In: *Experimental & Molecular Medicine* 50.8 (Aug. 2018), p. 96. ISSN: 2092-6413. DOI: 10.1038/s12276-018-0071-8. URL: https://doi.org/10.1038/s12276-018-0071-8.

[7] Grace X. Y. Zheng et al. "Massively parallel digital transcriptional profiling of single cells". In: *Nature Communications* 8.1 (Jan. 2017), p. 14049. ISSN: 2041-1723. DOI: 10.1038/ncomms14049. URL: https://doi.org/10.1038/ncomms14049.

[8] Malte D Luecken and Fabian J Theis. "Current best practices in single-cell RNA-seq analysis: a tutorial". In: *Molecular Systems Biology* 15.6 (2019), e8746. DOI: https://doi.org/10.15252/msb.20188746. eprint: https://www.embopress.org/doi/pdf/10.15252/msb.20188746. URL: https://www.embopress.org/doi/abs/10.15252/msb.20188746.

[9] Aisha A. AlJanahi, Mark Danielsen, and Cynthia E. Dunbar. "An Introduction to the Analysis of Single-Cell RNA-Sequencing Data". In: *Molecular Therapy - Methods & Clinical Development* 10 (2018), pp. 189–196. ISSN: 2329-0501. DOI: https://doi.org/10.1016/j.omtm.2018.07.003. URL: http://www.sciencedirect.com/science/article/pii/S2329050118300664.

[10] F. William Townes et al. "Feature selection and dimension reduction for single-cell RNA-Seq based on a multinomial model". In: *Genome Biology* 20.1 (Dec. 2019), p. 295. ISSN: 1474-760X. DOI: 10.1186/s13059-019-1861-6. URL: https://doi.org/10.1186/s13059-019-1861-6.

[11] Laurens van der Maaten and Geoffrey Hinton. "Visualizing Data using t-SNE". In: *Journal of Machine Learning Research* 9 (2008), pp. 2579–2605. URL: http://www.jmlr.org/papers/v9/vandermaaten08a.html.

[12] Leland McInnes, John Healy, and James Melville. *UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction*. 2020. arXiv: 1802.03426 [stat.ML].

[13] Vincent D. Blondel et al. "Fast unfolding of communities in large networks". In: *Journal of Statistical Mechanics: Theory and Experiment* P10008 (Oct. 2008), pp. 1–12. DOI: 10.1088/1742-5468/2008/10/P10008. URL: https://hal.archives-ouvertes.fr/hal-01146070.

[14] Frank Wilcoxon. "Individual Comparisons by Ranking Methods". In: *Biometrics Bulletin* 1.6 (1945), pp. 80–83. ISSN: 00994987. URL: http://www.jstor.org/stable/3001968.

[15] *Anaconda Software Distribution*. Version Vers. 2-2.4.0. Nov. 2016. URL: https://anaconda.com.

[16] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN: 1441412697.

[17] Peter V. Kharchenko, Lev Silberstein, and David T. Scadden. "Bayesian approach to single-cell differential expression analysis". In: *Nature Methods* 11.7 (July 2014), pp. 740–742. ISSN: 1548-7105. DOI: 10.1038/nmeth.2967. URL: https://doi.org/10.1038/nmeth.2967.

[18] Jean Fan et al. "Characterizing transcriptional heterogeneity through pathway and gene set overdispersion analysis". In: *Nature Methods* 13.3 (Mar. 2016), pp. 241–244. ISSN: 1548-7105. DOI: 10.1038/nmeth.3734. URL: https://doi.org/10.1038/nmeth.3734.

[19] Simon N. Wood. *Generalized Additive Models: An Introduction with R*. 2nd ed. Chapman and Hall/CRC, 2017.

[20] Gabor Csardi and Tamas Nepusz. "The igraph software package for complex network research". In: *InterJournal* Complex Systems (2006), p. 1695. URL: https://igraph.org.

[21] Thomas M. J. Fruchterman and Edward M. Reingold. "Graph drawing by force-directed placement". In: *Software: Practice and Experience* 21.11 (1991), pp. 1129–1164. DOI: https://doi.org/10.1002/spe.4380211102. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.4380211102. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.4380211102.

[22] Jingzhou Liu et al. "Visualizing Large-scale and High-dimensional Data". In: *Proceedings of the 25th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee. 2016, pp. 287–297.

[23] M. Ashburner et al. “Gene ontology: tool for the unification of biology. The Gene Ontology Consortium”. eng. In: *Nature genetics* 25.1 (May 2000). PMC3037419[pmcid], pp. 25–29. ISSN: 1061-4036. DOI: 10.1038/75556. URL: https://doi.org/10.1038/75556.

[24] The Gene Ontology Consortium. “The Gene Ontology resource: enriching a GOld mine”. In: *Nucleic Acids Research* 49.D1 (Dec. 2020), pp. D325–D334. ISSN: 0305-1048. DOI: 10.1093/nar/gkaa1113. eprint: https://academic.oup.com/nar/article-pdf/49/D1/D325/35364517/gkaa1113.pdf. URL: https://doi.org/10.1093/nar/gkaa1113.

[25] Christopher S. McGinnis, Lyndsay M. Murrow, and Zev J. Gartner. “DoubletFinder: Doublet Detection in Single-Cell RNA Sequencing Data Using Artificial Nearest Neighbors”. In: *Cell Systems* 8.4 (Apr. 2019), 329–337.e4. ISSN: 2405-4712. DOI: 10.1016/j.cels.2019.03.003. URL: https://doi.org/10.1016/j.cels.2019.03.003.

[26] Christoph Hafemeister and Rahul Satija. “Normalization and variance stabilization of single-cell RNA-seq data using regularized negative binomial regression”. In: *bioRxiv* (2019). DOI: 10.1101/576827. eprint: https://www.biorxiv.org/content/early/2019/03/14/576827.full.pdf. URL: https://www.biorxiv.org/content/early/2019/03/14/576827.

[27] John F. Ouyang. *ShinyCell: Shiny Interactive Web Apps for Single-Cell Data*. R package version 2.0.0. 2021. URL: https://github.com/SGDDNB/ShinyCell.

[28] 10x Genomics. *Datasets -Single Cell Gene Expression -Official 10x Genomics Support*. https://support.10xgenomics.com/single-cell-gene-expression/datasets. Accessed: 2021-04-21. 2021.

[29] 10x Genomics. *3k PBMCs from a Healthy Donor*. Single Cell Gene Expression Dataset by Cell Ranger 1.1.0. May 2016. URL: https://support.10xgenomics.com/single-cell-gene-expression/datasets/1.1.0/pbmc3k.

[30] 10x Genomics. *10k Peripheral blood mononuclear cells (PBMCs) from a healthy donor, Single Indexed*. Single Cell Gene Expression Dataset by Cell Ranger 4.0.0. July 2020. URL: https://support.10xgenomics.com/single-cell-gene-expression/datasets/4.0.0/SC3_v3_NextGem_SI_PBMC_10K.

[31] 10x Genomics. *33k PBMCs from a Healthy Donor*. Single Cell Gene Expression Dataset by Cell Ranger 1.1.0. Sept. 2016. URL: https://support.10xgenomics.com/single-cell-gene-expression/datasets/1.1.0/pbmc33k.

[32] William M. Rand. “Objective Criteria for the Evaluation of Clustering Methods”. In: *Journal of the American Statistical Association* 66.336 (1971), pp. 846–850. DOI: 10.1080/01621459.1971.10482356. eprint: https://www.tandfonline.com/doi/pdf/10.1080/01621459.1971.10482356. URL: https://www.tandfonline.com/doi/abs/10.1080/01621459.1971.10482356.

A Appendix

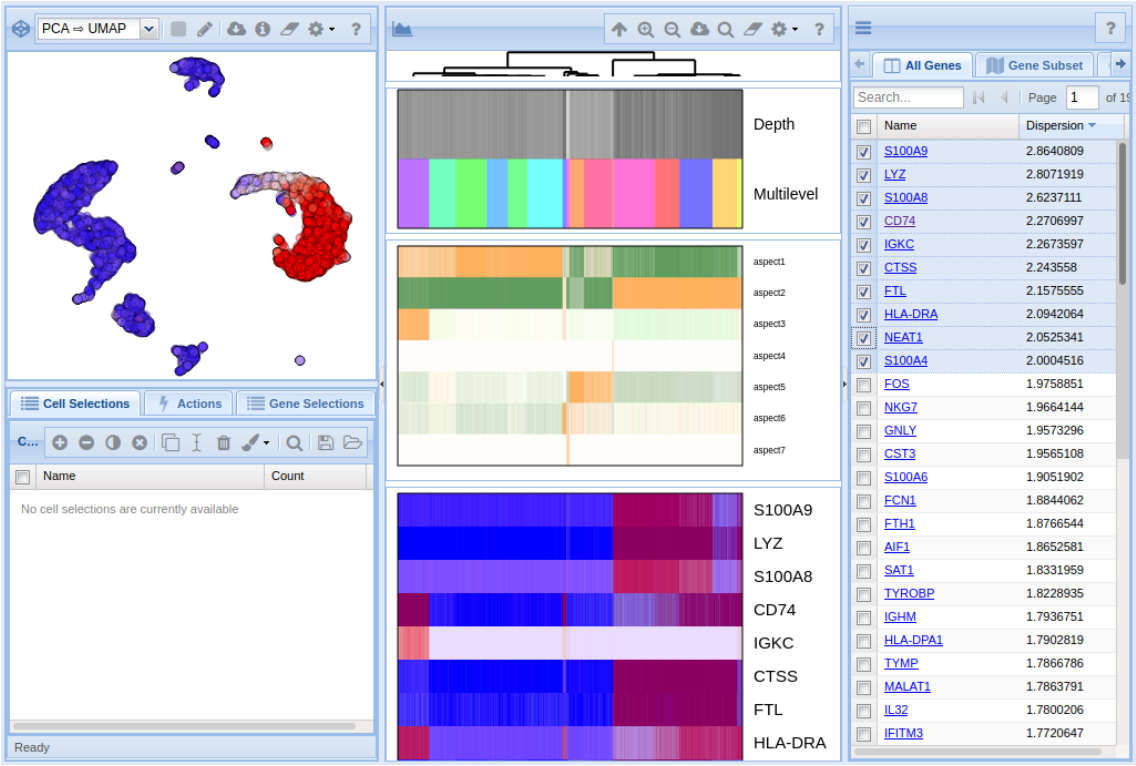


Figure 16: Web application of Pagoda2. On the left, the expression of the gene *S100A9* is highlighted in the clusters. Red indicates upregulation of the gene, while blue indicates downregulation.

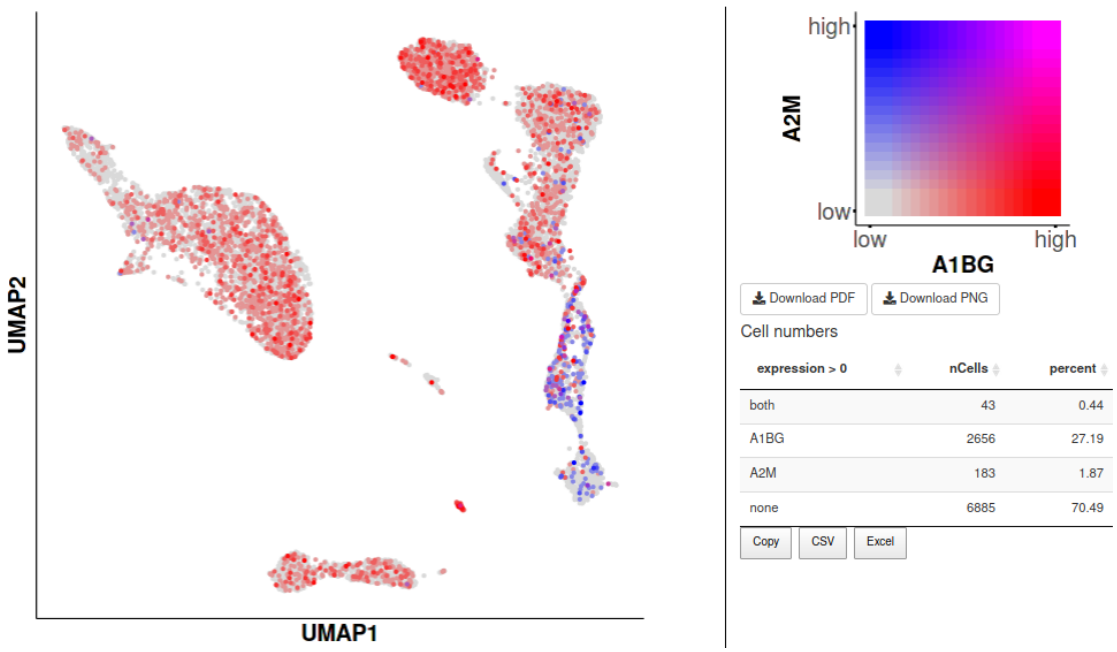


Figure 17: Section from the interactive web application of the Seurat-pipeline using ShinyCell. Here, the co-expression of the genes A2M and A1BG is highlighted in the UMAP embedding of the data.

Cluster ID	Unique Cells	Total Cells	Proportion (%)
0	9	1 836	0.5
1	0	1 409	0.0
2	31	1 245	2.5
3	0	1 213	0.0
4	0	1 164	0.0
5	1	807	0.1
6	5	485	1.0
7	0	417	0.0
8	0	392	0.0
9	11	359	3.1
10	30	145	20.7
11	0	133	0.0
12	24	93	25.8
13	3	69	4.3

Table 5: Distribution of cells kept only in the Seurat-pipeline across clusters.

Cluster ID	Unique Cells	Total Cells	Proportion (%)
1	30	718	4.2
2	21	430	4.9
3	42	722	5.8
4	70	158	44.3
5	0	20	0.0
6	2	906	0.2
7	3	578	0.5
8	6	781	0.8
9	0	1 033	0.0
10	1	624	0.2
11	42	109	38.5
12	15	973	1.5
13	41	927	4.4
14	4	93	4.3
15	181	1 229	14.7
16	29	839	3.5

Table 6: Distribution of cells kept only in the Pagoda2-pipeline across clusters.

IDs	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	552	-	-	14	-	-	-	-	-	-	-	214	-	-	1047	-
1	-	-	-	-	-	463	119	-	499	326	-	-	-	-	-	2
2	136	-	333	6	-	-	-	-	-	-	-	738	-	-	1	-
3	-	-	-	-	-	435	7	-	512	259	-	-	-	-	-	-
4	-	-	-	-	-	-	405	690	14	22	-	-	2	-	-	31
5	-	6	-	-	-	3	42	70	6	7	-	-	5	1	-	666
6	-	-	-	2	-	-	-	-	-	-	-	-	478	-	-	-
7	-	403	-	-	-	-	-	-	-	-	-	-	4	3	-	7
8	-	-	-	-	-	-	-	-	-	-	-	-	392	-	-	-
9	-	-	344	-	-	-	-	-	-	-	-	4	-	-	-	-
10	-	-	2	-	20	-	-	1	2	-	-	-	5	85	-	-
11	-	-	-	-	-	3	2	14	-	9	1	-	-	-	-	104
12	-	-	1	66	-	-	-	-	-	-	-	2	-	-	-	-
13	-	-	-	-	-	-	-	-	-	-	66	-	-	-	-	-

Table 7: Matching barcode names across the two clusterings. Therefore, barcodes that were only kept in one workflow after QC are not included. The rows are the clusters from Seurat and the columns the clusters from Pagoda2.